SEPRAN

SEPRA ANALYSIS

SEPRAN EXAMPLES

GUUS SEGAL

SEPRAN EXAMPLES

January 2013

Ingenieursbureau SEPRA
Park Nabij 3
2491 EG Den Haag
The Netherlands
Tel. 31 - 70 3871309

Contents

1 Introduction

In this manual we give a number of examples as illustration of how to use SEPRAN for specific problems.

In fact the subdivision of this manual is exactly the same as in the Standard Problems except for the first two chapters. So examples in for example Chapter 7 refer to elements introduced in the Standard Problems Manual Chapter 7. These examples must be seen as a supplement to the examples treated in the manual Standard Problems. In the rest of Chapter 1 we give some examples showing some specific items treated in the Users Manual and in Chapter 2 the same for items treated in the Programmers Guide.

2 Typical examples showing the use of the Programmers Guide

This chapter is under preparation.

3 Second order elliptic and parabolic equations

In this chapter we consider several types of elliptic and parabolic equations of second order.
The following Sections are available:

3.1 Second order real elliptic and parabolic equations with one-degree of freedom.
In this section the general second order quasi linear elliptic equation is treated. Due to the presence of a time derivative the corresponding parabolic equation is treated as well.
The number of unknowns per point is 1.

3.2 Second order complex elliptic and parabolic equations with one degree of freedom.
This section has the same purpose as Section 3.1, however, in this case complex unknowns are considered.

3.3 Non-linear equations.
This section is devoted to some special non-linear differential equations.

3.4 δ-type source terms.
This section treats a very special type of source term. It has no general character.

3.5 Second order real elliptic and parabolic equations with two degrees of freedom.
This section has the same purpose as Section 3.1, however, in this case the number of unknowns is equal to two per point.

3.6 Extended second order real linear elliptic and parabolic equations with two degrees of freedom
This section has the same purpose as Section 3.5, however extra terms defining the coupling between the equations are present.

3.1 Second order real linear elliptic and parabolic equations with one degree of freedom

In this section we treat the following examples of real elliptic and parabolic equations with one degree of freedom.

3.1.1 An artificial mathematical example

In this section we consider an artificial example of the solution of a Laplace equation with Neumann type boundary conditions. The purpose of this example is to show how the elements of this chapter may be used and how coefficients must be filled.
To get this example into your local directory use:

```
sepgetex exam3-1-1
```

and to run it use:

```
sepmesh exam3-1-1.msh
sepcomp exam3-1-1.prb
```

Consider the square Ω: (0,1) × (0,1) drawn in Figure 3.1.1.1.

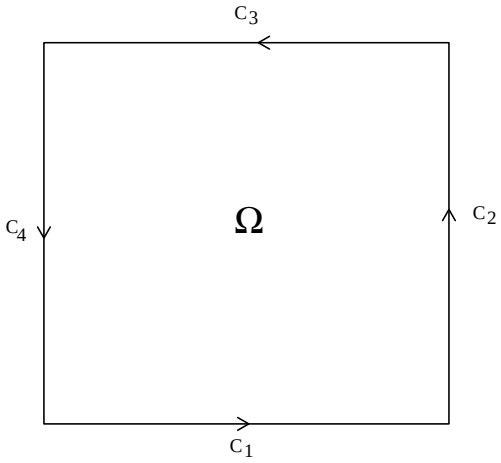


Figure 3.1.1.1: Definition of region for artificial mathematical example

We assume that we have to solve the Laplace equation:

$$-\Delta\phi = 0$$

In order to solve this equation it is necessary to impose boundary conditions at each side. In our example we define the following boundary conditions:

C1: $\phi = 0$
C2: $\frac{\partial\phi}{\partial n} = y$
C3: $\phi + \frac{\partial\phi}{\partial n} = 2x$
C4: $\phi = 0$

One easily verifies that the exact solution of this equation is given by $\phi = xy$

The region is subdivided into triangles by the submesh generator "RECTANGLE". As an example linear triangles have been used.
SEPMESH needs an input file.
This input file is standard and will not be repeated.
The input file for sepcomp uses laplace as type of equation. At the curves C2 and C3 we need boundary elements, since we are dealing with non-homogeneous natural boundary elements.
The potential at curves C1 and C4 is prescribed, hence we need essential boundary conditions at those curves.
Since we have different values for the natural boundary conditions at the curves C2 and C3 it is

necessary to use a coefficients block.
The values of the right-hand side functions for these boundary conditions are stored in the vectors h1 and h2 respectively.
The following input file may be used to solve the problem:

```
*****************************************************************************
*
*      File:  exam3-1-1.prb
*
*      Contents:  Input for program sepcomp described in Section 3.1.1 in
*                 the manual examples
*                 Artificial analytical example
*
*****************************************************************************
*

*  Problem definition

problem
   laplace                   # standard laplace problem
   boundary_elements
      belm1=curves(c2)          # natural boundary group 1 refers to c2
      belm2=curves(c3)          # natural boundary group 2 refers to c3
   essential_boundary_conditions
      curves (c1)
      curves (c4)
end

structure

  #  Define the structure of the matrix

   matrix_structure: symmetric       # the matrix is symmetrical

  #  Fill essential boundary conditions

   prescribe_boundary_conditions potential = 0

  # Build matrix and right-hand side and solve system of equations
  # We need vectors along c2 and c3 to define the functions
  # Since the boundary elements require different input at different
  # boundaries, we need to use the input block coefficients

   h1 = y_coor, curves (c2)
   h2 = 2*x_coor, curves (c3)

   solve_linear_system potential, seq_coef = 1
   print potential
   plot_contour potential
   plot_colored_levels potential

end

* Definition of coefficients
```

```
coefficients
   bngrp 1                    # First boundary group (curve 2)
      diff_flux = h1          # h = y
   bngrp 2                    # Second boundary group (curve 3)
      diff_sigma = 1          # sigma = 1
      diff_flux = h2          # h = 2x
end
end_of_sepran_input
```

Figure 3.1.1.2 shows the contour plot. This plot may be visualized by the program sepdisplay.

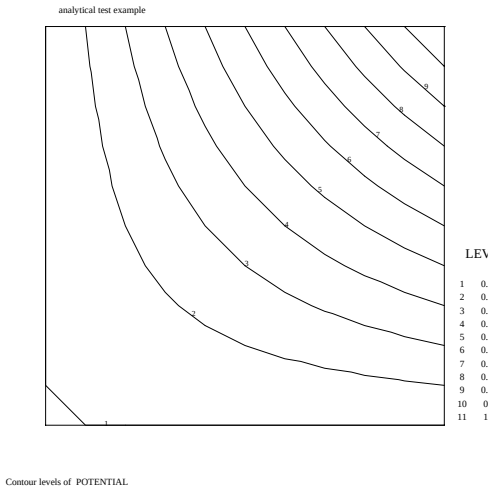


Figure 3.1.1.2: Contour plot

3.1.2 Propagation of concentration in a flow in a curved channel

In this section we consider the propagation of concentration in a flow in a curved channel.
To get this example into your local directory use:

```
sepgetex exam3-1-2a
```

and to run it use:

```
sepmesh exam3-1-2a.msh
sepcomp exam3-1-2a.prb
```

The region of definition is given in Figure 3.1.2.1. The cross-section in the x-y plane contains two

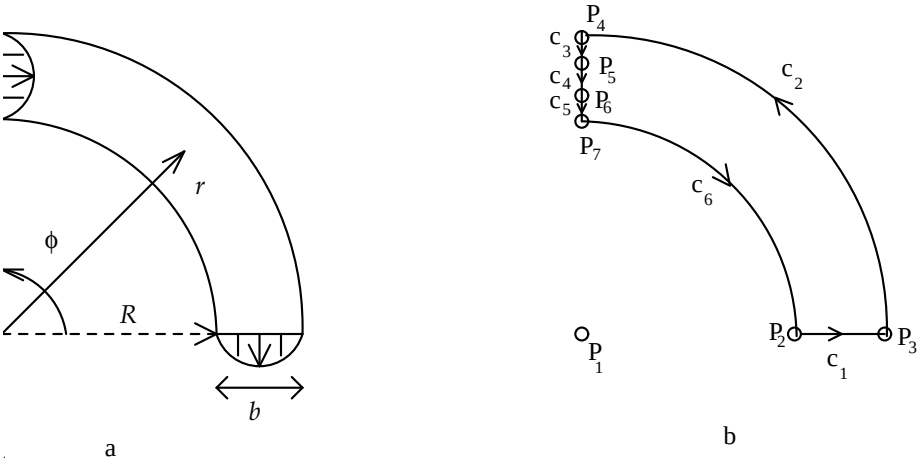


Figure 3.1.2.1: Curved channel. a) definition of region b) definition of curves

concentric arcs closed by straight lines. Through the channel we have a flow parallel to the arcs. The flow in radial direction is quadratic with maximum velocity one and zero at the circular walls. So the velocity can be described by the following formulae:

$$u_\phi = \frac{(r-R+b)(r-R)}{4}$$
$$u_1 = -\frac{u_\phi y}{r}$$
$$u_2 = \frac{u_\phi x}{r}$$

R denotes the radius of the inner circle, b denotes the width of the channel and r the radial distance from the origin. u_ϕ denotes the velocity in ϕ direction.
At the inflow a concentration of some quantity c is given. c is defined as follows:

$$\text{c} = 0 \text{ for } R \le y \le R + b/4 \text{ and } R + 3b/4 \le y \le R + b$$
$$\text{c} = 1 \text{ for } R + b/4 \le y \le R + 3b/4$$

At the outflow boundary we assume that the concentration is constant in normal direction, which means that we have the boundary condition:

$$\frac{\partial c}{\partial n} = 0$$

We assume that the circular walls are weakly permeable with respect to the concentration. This boundary condition may be described by

$$\frac{\partial c}{\partial n} + \sigma c = 0$$

The concentration c satisfies the convection-diffusion equation:

$$\mathbf{u} \cdot \nabla c - \text{ div } (\nu \nabla c) = 0$$

In our example we suppose that $R = 3$ and $b = 1$. The definition of the various curves and user points is given in Figure 3.1.2.1.

The region is subdivided into triangles by the submesh generator "GENERAL". As an example linear triangles have been used.
An example of an input file with respect to the mesh generator SEPMESH is given below:

```
*****************************************************************************
*
*      File:  exam3-1-2a.msh
*
*      Contents:  Mesh for the example 3-1-2 in the manual examples
*                 Propagation of concentration in a flow in a curved channel
*                 Coarseness of the grid defined by coarse
*                 The mesh is somewhat refined in the neighborhood of the
*                 two singular points P5 and P6
*****************************************************************************
*
constants
   reals
      radius = 3
      b = 1
end
mesh2d
   coarse(unit=.1)
   points
      p1 = (0,0,1)
      p2 = (radius,0,1)
      p3 = (radius+b,0,1)
      p4 = (0,radius+b,1)
      p5 = (0,radius+0.75*b,.5)
      p6 = (0,radius+0.25*b,.5)
      p7 = (0,radius,1)
   curves
      c1 = line ( p2,p3 )
      c2 = arc ( p3,p4,p1 )
      c3 = line ( p4,p5 )
      c4 = line ( p5,p6 )
      c5 = line ( p6,p7 )
      c6 = arc ( p7,p2,-p1 )
   surfaces
      s1 = general ( c1,c2,c3,c4,c5,c6)
   plot
end
```

Figure 3.1.2.2 shows the mesh generated by SEPMESH.

The internal elements are of type convection diffusion.

They require the parameters diffusion and velocity as input. In order to plot the velocity vectors we have chosen to create a vector u and a vector v, each consisting of one component per point. The velocity vector is created by `velocity = (u,v)` , which makes it a vector with two components per node. The boundary conditions at curves C3 to C5 are essential boundary conditions, the boundary conditions at curve C1 are natural boundary conditions requiring no special condition

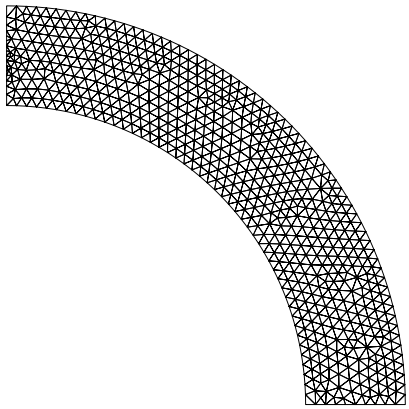


Figure 3.1.2.2: Plot of mesh generated by SEPMESH

and the boundary conditions at curves C2 and C6 are natural boundary conditions. For these boundaries it is sufficient to give coefficient σ) by `diff_sigma`.

For out specific example we use the following coefficients:

$\nu = 0.005$
$\sigma = 0.01$

The following input file may be used to solve the problem:

```
*****************************************************************************
*
*      File:  exam3-1-2a.prb
*
*      Contents:  Input for program sepcomp described in section 3-1-2 in
*                 the manual examples
*                 Propagation of concentration in a flow in a curved channel
*                 The standard sepcomp approach is used
*
*****************************************************************************
*
constants
   reals
      radius = 3
      b = 1
      diffusion = 0.005
      diff_sigma = 0.01
end
```

```
*  Problem definition
*

problem
   convection_diffusion
   boundary_elements
      belm1 = curves ( c2 )
      belm2 = curves ( c6 )
   essential_boundary_conditions
      curves ( c3 to c5 )
end

structure

  #  Fill essential boundary conditions

   prescribe_boundary_conditions concentration = 1, curves(c4)

  # Build matrix and right-hand side and solve system of equations

   r = sqrt(x_coor^2+y_coor^2)
   uphi = 0.25*(r-(Radius+b))*(r-Radius)
   u = -uphi*y_coor/r
   v =  uphi*x_coor/r

   velocity = (u,v)
   plot_vector velocity

   solve_linear_system concentration
   print concentration
   plot_contour concentration
   plot_colored_levels concentration

end
end_of_sepran_input
```

Figure 3.1.2.3 shows the contour plot. This plot may be visualized by the program SEPDISPLAY.

If we want to compute the same problem with a very small diffusion term ($\nu = 0.00005$). The input files in this case are called `exam3-1-2b.msh` and `exam3-1-2b.prb`. You get them in your local directory by

```
sepgetex exam3-1-2b
```

Figure 3.1.2.4 shows the contour plot.

In order to get a slightly smoother plot upwind may be applied.

Use `exam3-1-2c` for this case. Figure 3.1.2.5 shows the contour plot. Due to the discontinuities of the concentration at inflow a complete smooth contour is not possible.

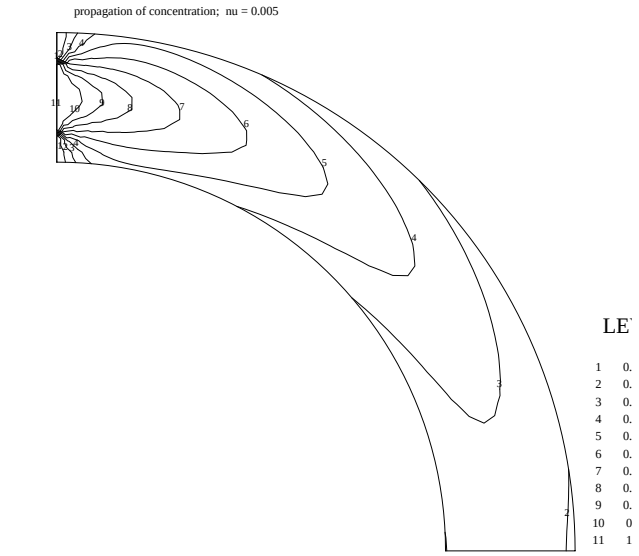


Figure 3.1.2.3: Contour plot

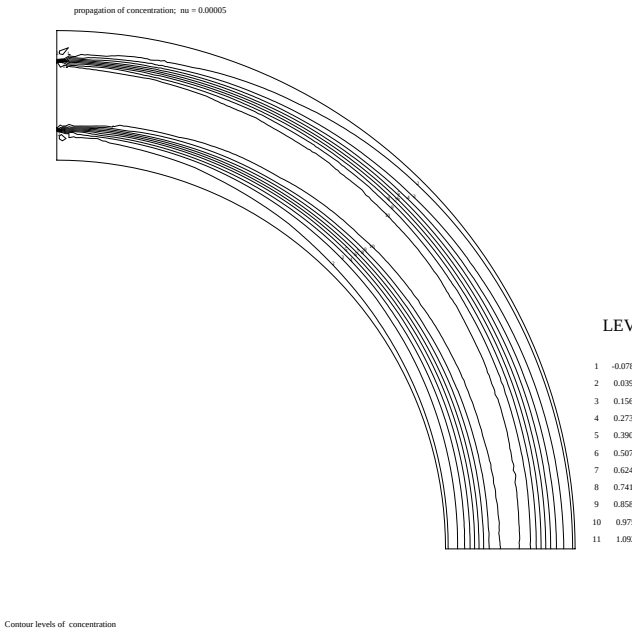


Figure 3.1.2.4: Contour plot with small value of ν

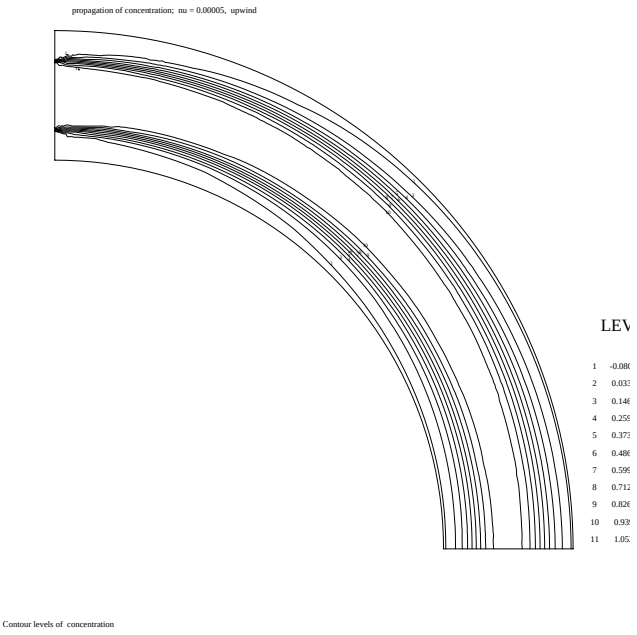


Figure 3.1.2.5: Contour plot with small value of ν and upwind

3.1.3 An example of a simple heat equation

In this section we consider exactly the same problem as in Section 6.4.1 of the Users Manual. The only difference is that in each time step we want to compute the gradient of the temperature and also some other special quantities (see below).
In order to get this example in your local directory use the command:

```
sepgetex heatequ4
```

You can run the example by performing the following steps:

```
sepmesh heatequ4.msh
view the mesh for example by:  sepview sepplot.001 or sepdisplay
seplink heatequ4
heatequ4 < heatequ4.prb
seppost heatequ4.pst
view the plots for example by:  sepview sepplot.001 or sepdisplay
```

Consider the heat equation

$$\frac{\partial T}{\partial t} - 0.5\Delta T = 0 \tag{3.1.3.1}$$

with Δ the Laplacian operator. We assume that the region at which this equation is defined is the unit square $(0,1) \times (0,1)$.
We suppose that the initial condition is given by

$T(\mathbf{x}, \mathbf{0}) = \mathbf{sin}(\mathbf{x})\mathbf{sin}(\mathbf{y})$

and the boundary conditions by

$T(\mathbf{x}, \mathbf{t}) = \mathbf{sin}(\mathbf{x})\mathbf{sin}(\mathbf{y})\mathbf{exp}(-\mathbf{t})$ at all four boundaries.

It is easy to verify that the exact solution in this case is also equal to

$T(\mathbf{x}, \mathbf{t}) = \mathbf{sin}(\mathbf{x})\mathbf{sin}(\mathbf{y})\mathbf{exp}(-\mathbf{t})$

In order to solve this problem a mesh is created by sepmesh using the submesh generator general. An example input file for sepmesh is the file heatequ4.msh:

```
* file heateq4.msh
*
* mesh for the unit square (0,1) x (0,1)
mesh2d
   coarse(unit=0.1)
   points
      p1=(0,0,1)
      p2=(1,0,1)
      p3=(1,1,1)
      p4=(0,1,1)
   curves
      c1=cline1(p1,p2)
      c2=cline1(p2,p3)
      c3=cline1(p3,p4)
      c4=cline1(p4,p1)
   surfaces
      s1=general3(c1,c2,c3,c4)
   plot (jmark=5, numsub=1)
end
```

Since the initial and boundary conditions are space and time dependent it is necessary to provide user written function subroutines.
The main program may have the following shape (file heatequ4.f)

```
      program heatequation_4
      implicit none
      call sepcom(0)
      end

! ********************************************************************
!
!    function func for the initial condition
!    contains also the exact solution
!
! ********************************************************************
      function func ( ichoice, x, y, z )
      implicit none
      double precision func, x, y, z
      integer ichoice
      double precision t, tout, tstep, tend, t0, rtimdu
      integer iflag, icons, itimdu
      common /ctimen/ t, tout, tstep, tend, t0, rtimdu(5), iflag,
     +               icons, itimdu(8)

      func = exp(-t)*sin(x)*sin(y)

      end
! ********************************************************************
!
!    function for essential boundary conditions
!
! ********************************************************************
      function funcbc ( ichoice, x, y, z )
      implicit none
      double precision funcbc, x, y, z
      integer ichoice
      double precision t, tout, tstep, tend, t0, rtimdu
      integer iflag, icons, itimdu
      common /ctimen/ t, tout, tstep, tend, t0, rtimdu(5), iflag,
     +               icons, itimdu(8)

      if ( ichoice.eq.1 ) then
         funcbc = sin(x)*sin(y)*exp(-t)
      else if ( ichoice.eq.2 ) then
         funcbc = sin(x)*sin(y)*exp(-t)
      else if ( ichoice.eq.3 ) then
         funcbc = sin(x)*sin(y)*exp(-t)
      else if ( ichoice.eq.4 ) then
         funcbc = sin(x)*sin(y)*exp(-t)
      end if

      end
```

In this example we want to perform some extra actions compared to the standard solution of a time-dependent problem. For that reason we need an input block `structure` in the input file. The structure of the main program consists of the following steps:

- Create initial solution
- Solve heat equation (time-dependent)
- Create exact solution
- Compute and print error at the last time-step (i. e. t=1)
- Compute and print the gradient of the temperature at the last time-step
- Compute and print the volume integral of the temperature at the last time-step
- Compute and print the boundary integral over curve c2 of the temperature at the last time-step
- Write the final solution and gradient to the file sepcomp.out for postprocessing purposes. This last step is superfluous since in each time-step the result is written.

The following input file may be used as input for heatequ4:

```
*  file: heatequ4.prb
*
*  problem definition for time-dependent heat equation
*  linear triangles  type number 800
*
set warn off  ! suppress warnings

constants           # See Users Manual Section 1.4
   vector_names
      temperature
      exact_temperature
      temperature_grad
   variables
      error
      temp_int
      int_temp_boun
end

problem
   types
      elgrp1 = 800            # Standard general second order parabolic equation
   essbouncond
      curves(c1,c4)          # Temperature given at all sides
end

*  Definition of matrix structure

matrix
   symmetric
end

* Definition of structure of the program

structure
   create_vector, temperature         # start vector (t=0)
   solve_time_dependent_problem
   create_vector, exact_temperature         # exact solution (t=1)
   error = norm_dif=3,vector1=temperature, vector2=exact_temperature
```

```
   print error, text = 'difference at time = 1'

   derivatives, seq_coef = 1, temperature_grad              # grad(T) (t=1)
   print temperature_grad

*   Integral of the temperature over the whole region

   temp_int = integral( seq_coef = 2, seq_integral = 1, temperature )

*   Integral of the temperature over curve c2

   boundary_integral, temperature, int_temp_boun

   print temp_int, text = 'Volume integral of the temperature'
   print int_temp_boun, text = 'Integral of the temperature over curve c2'
   output
end
*
*  Define initial conditions
*
create vector
   func = 1                       # The initial condition is given in FUNCCF
end
*
*  Essential boundary conditions
*
essential boundary conditions
   curves(c1,c4),(func=1)      # The boundary conditions are given in FUNCBC
end
*
*  Definition of coefficients for the heat equation (t=0 only)
*
coefficients, sequence_number = 1
   elgrp1(nparm=20)
      coef6  = 0.5                # a11 = 0.5
      coef9  = coef 6             # a22 = 0.5
      coef17 = 1                  # rho = 1
end
*
*  Definition of the coefficient for the volume integration
*
coefficients, sequence_number = 2
   elgrp1(nparm=10)
      coef4  = 1                  # f = 1
end
*
derivatives
   icheld = 6      # a * grad T = heat-flux
end

# Definition of integral to be computed

integrals
   icheli = 2                      # / fT d omega
end
```

```
# Definition of boundary integral to be computed

boundary_integral
   ichint = 1                      # / fT d gamma
   ichfun = 0                      # f = 1
   curves(c2)                      # integration over C2
end

output
   v1 = icheld=6, seq_coefficients=1   # a * grad T = heat-flux
                                       # It is necessary to give the coefficient
                                       # sequence number, since output at t=0
                                       # is produced before the system of
                                       # equations is build.
end

# Definition of time integration

time_integration, sequence_number = 1
   method = crank_nicolson             # Second order accurate in time
   tinit = 0
   tend  = 1
   tstep = 0.1
   toutinit = 0
   toutend  = 1
   toutstep = 0.1                      # In each time step the result is written
   seq_boundary_conditions = 1
   seq_coefficients = 1
   diagonal_mass_matrix
   stiffness_matrix = constant
   mass_matrix = constant
   right_hand_side = zero             # There is no right-hand side contribution
                                      # of source terms and natural bc's
end
```

Mark that in the input block for the time integration we use the fact that the coefficients of both matrices do not depend on time. Hence both matrices remain constant.
Since there is no source term, and there are no natural boundary conditions with non-zero right-hand side, we may use the option `right_hand_side = zero`. The only reason that we have a non-zero right-hand side in the system of equations to be solved in the time integration is due to the previous time step and also to the essential boundary conditions.
For linear triangles a lumped mass matrix is accurate enough and for that reason we use `diagonal_mass_matrix`.
In each time step the results are written for postprocessing.

In the input block `output` we also compute the gradient of the temperature multiplied by the coefficient of the second order term. This requires the same coefficients as for the building of the stiffness matrix. Since we want to produce output even at $t = 0$, it is necessary to give explicitly the sequence number of the input block `coefficients` for the derivatives. First the derivatives are computed and written to the file, and then the stiffness matrix is built.

The solution may be visualized by seppost using the file heatequ4.pst as input file:

```
* file: heatequ4.pst
*
* input for seppost
```

```
*
set warn off  ! suppress warnings

postprocessing
   time = (0,1)
      plot contour temperature, minlevel = 0, maxlevel = 1
      plot vector temperature_grad, factor = 0.5
   time history plot point(.5,.5) temperature, scales(0,1,0,0.25)//
      number format = (1,1,1,3)
end
```

3.1.4 An artificial example of the use of the membrane boundary condition

In this section we consider an artificial example of the use of boundary conditions of type 6. This boundary condition allows for a jump in the solution and is used to simulate a membrane. To get this example in your local directory use the command:

```
sepgetex interf
```

To run the example use the commands:

```
sepmesh interf.msh
view the plots
seplink interf
interf < interf.prb
seppost interf.pst
view the plots
```

Consider the region drawn in Figure 3.1.4.1.

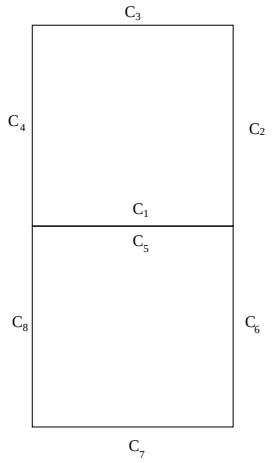


Figure 3.1.4.1: Definition of region for membrane boundary condition

This region consists of the squares (0,1) × (0,1) and (0,1) × (0,-1) separated by a membrane at $y = 0$. We assume that in both squares we have to solve the Laplace equation:

$$-\Delta p = 0$$

The following boundary conditions will be used:

C2,C3,C4: $p = 1 - y$
C6,C7,C8: $p = 2 + y$

At the membrane we impose the "jump" condition:

$$\sigma(p_u - p_l) + \frac{\partial p}{\partial n} = h \tag{3.1.4.1}$$

If we set: $\sigma = -1$ and $h = 2$ then one easily verifies that the exact solution of this equation is given by $p = 1 - y$ for $y > 0$ and $p = 2 + y$ for $y < 0$.
At $y = 0$ p has the value 1 for the upper region and 2 for the lower region, which implies that p is discontinuous.

In order to impose the membrane boundary condition it is necessary that the curves C1 and C5 are strictly disjoint. In this way we get two sets of disjoint points each of which representing a different value for p. The coordinates of the curves C1 and C5, however, are identical. In order to connect the curves C1 and C5 connection elements are used. These elements consist of a linear element at C1 connected to the corresponding linear element at C5 and hence may be considered as quadrilateral elements with thickness zero.
In our example we use linear triangles in each rectangle and linear connection elements at the membrane.
An example of an input file for SEPMESH is given below:

```
#  interf.msh
#
#  mesh file for 2d membrane example
#  See Manual Standard Elements Section 3.1.4
#
#  To run this file use:
#     sepmesh interf.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      x_left = 0        # x-coordinate of left-hand side
      x_right = 1       # x-coordinate of right-hand side
      y_bottom = -1     # y-coordinate of bottom
      y_middle = 0      # y-coordinate of membrane
      y_top = 1         # y-coordinate of top
   integers
      n_horizontal = 5  # number of elements in horizontal direction
      n_vertical = 5    # number of elements in vertical direction per surface
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2

      p1=( x_left,  y_middle)    # left-hand point of membrane in upper surface
      p2=( x_right, y_middle)    # right-hand point of membrane in upper surface
      p3=( x_right, y_top   )    # right-hand point on top boundary
      p4=( x_left,  y_top   )    # left-hand point on top boundary
      p5=( x_left,  y_middle)    # left-hand point of membrane in lower surface
      p6=( x_right, y_middle)    # right-hand point of membrane in lower surface
      p7=( x_right, y_bottom)    # right-hand point on bottom boundary
      p8=( x_left,  y_bottom)    # left-hand point on bottom boundary
#
#  curves
#
   curves           # See Users Manual Section 2.3

      c1 = line(p1,p2,nelm= n_horizontal)   # membrane curve in upper surface
      c2 = line(p2,p3,nelm= n_vertical)     # right-hand curve in upper surface
      c3 = line(p3,p4,nelm= n_horizontal)   # top curve in upper surface
      c4 = line(p4,p1,nelm= n_vertical)     # left-hand curve in upper surface
      c5 = line(p5,p6,nelm= n_horizontal)   # membrane curve in lower surface
      c6 = line(p6,p7,nelm= n_vertical)     # right-hand curve in lower surface
      c7 = line(p7,p8,nelm= n_horizontal)   # bottom curve in lower surface
      c8 = line(p8,p5,nelm= n_vertical)     # left-hand curve in lower surface
#
#  surfaces
```

```
#
   surfaces         # See Users Manual Section 2.4

      s1 = rectangle3 (c1,c2,c3,c4)    # upper surface
      s2 = rectangle3 (c5,c6,c7,c8)    # lower surface
#
#  Connect surfaces to element groups
#
   meshsurf
      selm1 = s1    # element group 1: upper surface
      selm2 = s2    # element group 2: lower surface
#
#  Define connection elements
#
   meshconnect
      celm3 = curves1(c1,c5)    # element group 3: connection elements
                                # from c1 to c5

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2
end
```

The internal elements are defined by type number 800. Only the coefficients 6 and 9 have to be defined; they get the value 1.
The boundary conditions at sides C2 to C4 and C6 to C8 are essential boundary conditions, the boundary conditions at sides C1 and C5 are the special membrane boundary conditions given by type number 804. Both σ and h must be defined for these elements.

Since in this case it is necessary to define a function subroutine for the essential boundary conditions, it is not possible to use the standard program SEPCOMP. Therefore we give the program interf based upon sepcomp and extended with the function subroutine FUNCBC.

```
************************************************************************
*
*      File:  interf.f
*
*      Contents:  Program for the test example
*                 in the SEPRAN manual Standard Problems Section 3.1.4
*
*      Usage:    Compile and link this program with the SEPRAN libraries
*                seplink interf
*                Run this program with input interf.prb
*                interf < interf.prb
*
************************************************************************
*
      program interf

!     --- example program for the interface boundary condition

      call sepcom ( 0 )

      end
```

```
!     --- Define essential boundary conditions as function of the coordinates

      function funcbc ( ichoice, x, y, z )
      implicit none
      double precision funcbc, x, y, z
      integer ichoice
      if ( ichoice==1 ) then

!     --- ichoice = 1, upper surface, p = 1-y

         funcbc = 1-y

      else

!     --- ichoice = 2, lower surface, p = 2+y

         funcbc = 2+y

      end if
      end
```

This program needs an input file which is the same as for SEPCOMP. The following input file may be used to solve the problem:

```
# interf.prb
#
#  problem file for 2d membrane example
#  See Manual Standard Elements Section 3.1.4
#
#  To run this file use:
#     sepcomp interf.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      kappa  = 1       # diffusion parameter
      sigma  = -1      # Parameter sigma for membrane boundary condition
      h      = 2       # Parameter h for membrane boundary condition
   vector_names
      pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1 = (type=800)      # Type number for second order equation
```

```
      elgrp2 = (type=800)      # Type number for second order equation
      elgrp3 = (type=804)      # Type number for membrane boundary condition
                               # See Standard problems Section 3.1
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2

      curves(c2 to c4)         # essential boundary conditions on c2 to c4
      curves(c6 to c8)         # essential boundary conditions on c6 to c8
end

#  Define essential boundary conditions
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c2 to c4), func=1    # The boundary conditions depend on y
   curves(c6 to c8), func=2    # so a function is needed

end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients
   elgrp1 (nparm=20)     # The coefficients are defined by 20 parameters
      coef 6 = (value= kappa)     # diffusion coefficient
      coef 9 = coef 6             # in upper surface
   elgrp2 (nparm=20)
      coef 6 = (value= kappa)     # diffusion coefficient
      coef 9 = coef 6             # in lower surface
   elgrp3 (nparm=15)     # The natural boundary conditions require 2 parameters
      coef 6 = (value= sigma)     # sigma
      coef 7 = (value= h)         # h
end

end_of_sepran_input
```

Once the solution has been computed, it may be printed and plotted by the postprocessing program SEPPOST. SEPPOST also requires an input file. The following input file prints the computed solution, makes a standard contour plot as well as a coloured contour plot.

```
#  interf.pst
#  Input file for postprocessing for 2d membrane example
#  See Manual Standard Elements Section 3.1.4
#
#
#  To run this file use:
#     seppost interf.pst > interf.out
#
#  Reads the files meshoutput and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2
```

```
#  Print the pressure
#  See Users Manual Section 5.3

   print pressure

#  Plot the pressure
#  See Users Manual Section 5.4

   plot contour pressure
   plot coloured contour pressure
end
```

Figures 3.1.4.2 shows the contour plot of the pressure and Figure 3.1.4.3 the coloured contour plot.

3.1.5 Cooling with convective heat-transfer at the boundaries

In this section we consider the problem, that a material at high temperature has to be cooled down. We assume, that the problem is two dimensional and that the material-cross-section has the shape of a rectangle (0.1 m x 0.05 m) with four cooled boundaries. As the cross-section is symmetrical, only the fourth part (a rectangle of 0.05 m x 0.025 m) has to be considered with two cooled boundaries and two boundaries with the boundary-condition "$\frac{\partial T}{\partial n} = 0$" (symmetry-boundary-condition), which in SEPRAN is satisfied automatically by not prescribing anything.
This example has been generated by Roman Denzin of the technical university of Darmstadt.
Consider the heat-equation:

$$c_p \rho \frac{\partial T}{\partial t} - \lambda \nabla T = 0 \tag{3.1.5.1}$$

with
c_p = heat-capacity of the material = 2000 J/(kg K),
ρ = density of the material = 1000 kg/m^3,
$c_p\rho$ is coef 17 of the element of type 800.
λ = heat-conductivity = 0.5 W/(m K)
coef6 respectively coef 9 of the element of type 800
The initial-condition is: T(x,y,t=0) = 200 degrees C
A common boundary condition of cooling- or heating-problems is a convective heat-transfer from the material to a surrounding fluid, which has a constant temperature at sufficient distance to the boundary. The specific heat-flux from the material to the fluid is given as:

$$q = \alpha(T_b - T_0) \tag{3.1.5.2}$$

with
α = surface-heat-transfer coefficient = 15 W(m^2 K),
T_b = temperature at the boundary [degrees C],
T_0 = temperature of the fluid at sufficient distance to the boundary = 5 degrees C.
The heat-flux from the inner of the material across the boundary is given as:

$$q = -\lambda \nabla T_b \tag{3.1.5.3}$$

with
λ = heat-conductivity of the material,
∇T_b = gradient of temperature at the boundary.
As these two heat-fluxes have to be equal, the boundary-condition is:

$$-\lambda \nabla T_b = \alpha(T_b - T_0), \tag{3.1.5.4}$$

hence

$$\lambda \nabla T_b + \alpha\ T_b = \alpha T_0. \tag{3.1.5.5}$$

To implement this in SEPRAN, boundary-elements of type 2 have to be used:

$$\alpha_{ij} \frac{\partial c}{\partial n} + \sigma c = h. \tag{3.1.5.6}$$

If you compare this equation with the boundary-condition above, you can see (with c replaced by T respectively T_b) that the coefficients of the boundary-elements of type 2 have to be defined as follows:
α_8 = coef6 (λ_x of the material)
α_{11} = coef9 (λ_y of the material)
σ = coef6 (α)
h = coef7 (αT_0)
(If coef 6 and coef 9 are omitted, these coefficients are taken from the input-block for the coefficients of the heat-equation, which is correct as well.)
In order to get this example in your local directory use the command:

```
sepgetex heatequ5
```

You can run the example by performing the following steps:

```
   sepmesh heatequ5.msh
   view the mesh for example by:  sepview sepplot.001 or sepdisplay
   sepcomp < heatequ5.prb
   seppost heatequ5.pst
   view the plots for example by:  sepview sepplot.001 or sepdisplay
```

In order to solve this problem a mesh is created by sepmesh using the submesh generator rectangle. An example input file for sepmesh is the following file:

```
*  file: heatequ5.msh
*
constants
   integers
      nelm1=20
      nelm2=40
end

mesh2d
   points
      p1=(0     , 0     )
      p2=(0.050 , 0     )
      p3=(0.050 , 0.025 )
      p4=(0     , 0.025 )
   curves
      c1=line2(p1,p2,nelm= nelm2)
      c2=line2(p2,p3,nelm= nelm1)
      c3=line2(p3,p4,nelm= nelm2)
      c4=line2(p4,p1,nelm= nelm1)
   surfaces
      s1=rectangle4(c1,c2,c3,c4)
   plot ( plotfm=10 )
end
```

In this example we are solving a standard heat equation and we do not require any extras from program sepcomp. For that reason it is sufficient to call program sepcomp with a standard input file. No input block `structure` is necessary.
The following input file may be used as input for sepcomp:

```
*  file: heatequ5.prb
*
*  problem definition for time-dependent heat equation
*  linear triangles  type number 800

set warn off  ! suppress warnings

constants           # See Users Manual Section 1.4
   vector_names
      temperature
end


problem
   types
```

```
      elgrp1 = 800        # Standard heat equation
natbouncond
   bngrp1 = (type=801)    # Boundary condition of type 2
   bngrp2 = (type=801)    # Boundary condition of type 2
bounelements
   belm1 = curves(c2)     # Boundary elements along curve c2
   belm2 = curves(c3)     # Boundary elements along curve c3
end
*
*  Definition of matrix structure
*
matrix
   symmetric
end
*
*  Define initial conditions
*
create vector
   value = 200       #T(t=0) = 200 degrees C
end

*
*  Definition of coefficients for the heat equation
*  and boundary conditions
*
coefficients

*  Definition of coefficients for the heat equation
   elgrp1(nparm=20)
      coef6  = (value=0.5)        # Lambda_x
      coef9  = coef 6             # Lambda_y
      coef17 = (value=2d6)        # cp*rho

*  Definition of coefficients for the boundary conditions
   bngrp1 (nparm=11)
      icoef 1 = 2             # Boundary conditions of type 2 (Default)
      coef 6 = (value=15)     # alpha
      coef 7 = (value=75)     # alpha * t_0

   bngrp2 (nparm=11)
      icoef 1 = 2             # Boundary conditions of type 2 (Default)
      coef 6 = (value=15)     # alpha
      coef 7 = (value=75)     # alpha * t_0
end

time_integration, sequence_number = 1
   method = euler_implicit              # time integration method
   tinit = 0
   tend  = 2000
   tstep = 50
   toutinit = 0
   toutend  = 2000
   toutstep = 400
   seq_boundary_conditions = 1
   seq_coefficients = 1
```

```
   diagonal_mass_matrix
   stiffness_matrix = constant          # the stiffness matrix does not depend
                                        # on time
   mass_matrix = constant               # the mass matrix does not depend
                                        # on time
   right_hand_side = constant           # the right-hand side does not depend
                                        # on time
                                        # It is not zero since the natural
                                        # boundary conditions contain a
                                        # contribution for the rhs
   print_time_history = ((0,0))
end
```

Mark that in the input block for the time integration we use the fact that the coefficients of both matrices do not depend on time. Hence both matrices remain constant.
Also the right-hand-side vector is constant. This vector is not zero, since the natural boundary condition has a non-zero right-hand side αT_0. The solution may be visualized by seppost using the file heatequ5.pst as input file:

```
* file heatequ5.pst
*
* input for seppost
*
set warn off  ! suppress warnings

postprocessing
  define plot parameters =  height=0.5
  plot identification, text='Cooling with convective heat-transfer'//
     origin=(3,19)
  time = (0, 2000)

*   Temperature at line y=0

    open plot
       compute temp_intersect = intersection temperature origin=(0,0)
       plot function temp_intersect, scales=(0,0.05,0,200), textx = 'x-coordinate [m]' //
           texty = 'Temperature [degree C]',  number format=(1,3,3,0)
    close plot

*   Temperature-distribution

    plot coloured levels temperature, nlevel=22, minlevel = 0, maxlevel = 200 //
          (yfact=1,plotfm=15), plot_legenda

    plot contour temperature, nlevel=21, minlevel = 0, maxlevel = 200

*   time history of temperature at (0,0)

   time history plot point(0,0) temperature, scales (0, 1800, 0, 200), //
     number format=(4,0,3,0), textx='Time [s]', texty='temperature [degree C]'
   time history plot max temperature, scales (0, 1800, 0, 200), //
     number format=(4,0,3,0), textx='Time [s]', texty='temperature [degree C]'

*   Time history of minimum and maximum
```

```
   time history print max temperature
   time history print min temperature

end
```

Figure 3.1.5.1 shows the temperature at the line $y = 0$ for the time levels 0 to 2000 seconds with steps of 400 seconds.

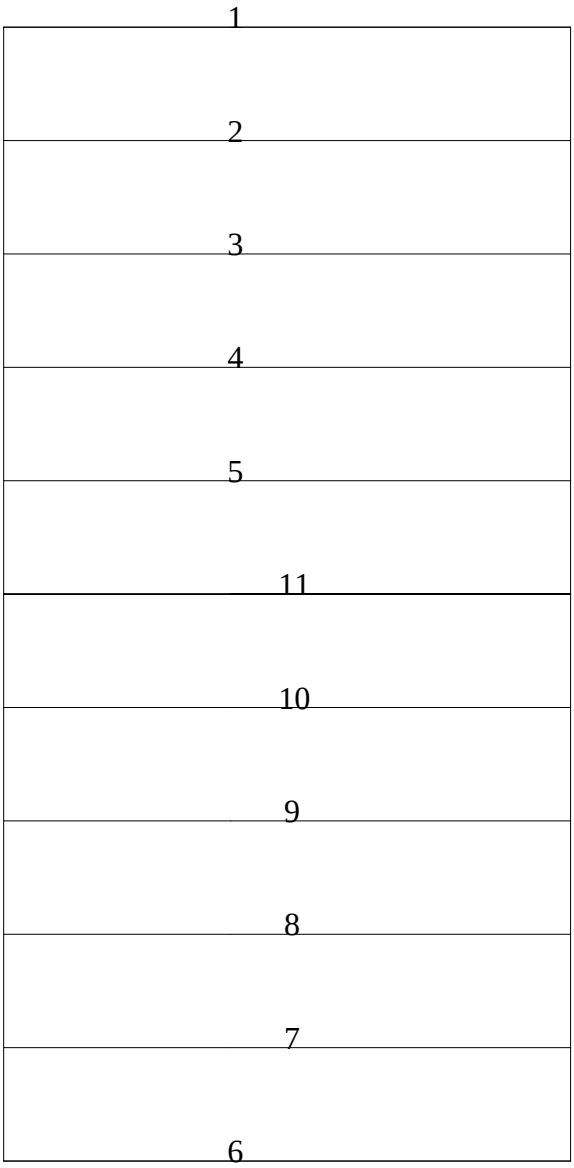
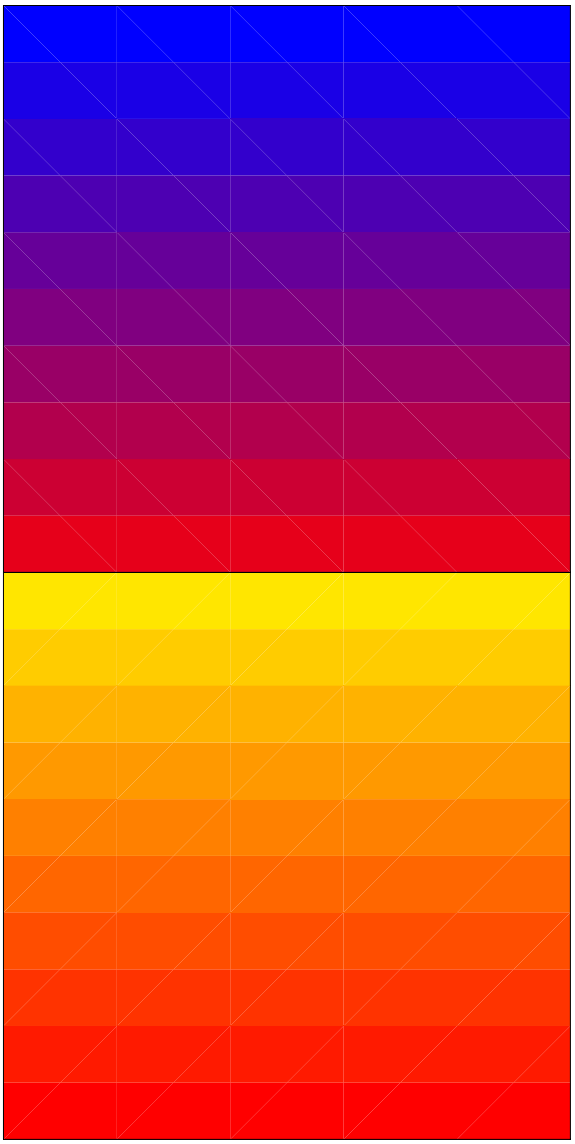


Figure 3.1.4.2: Isobars (computed pressure)



[height=7cm]

Figure 3.1.4.3: Coloured levels of pressure

[height=7cm]

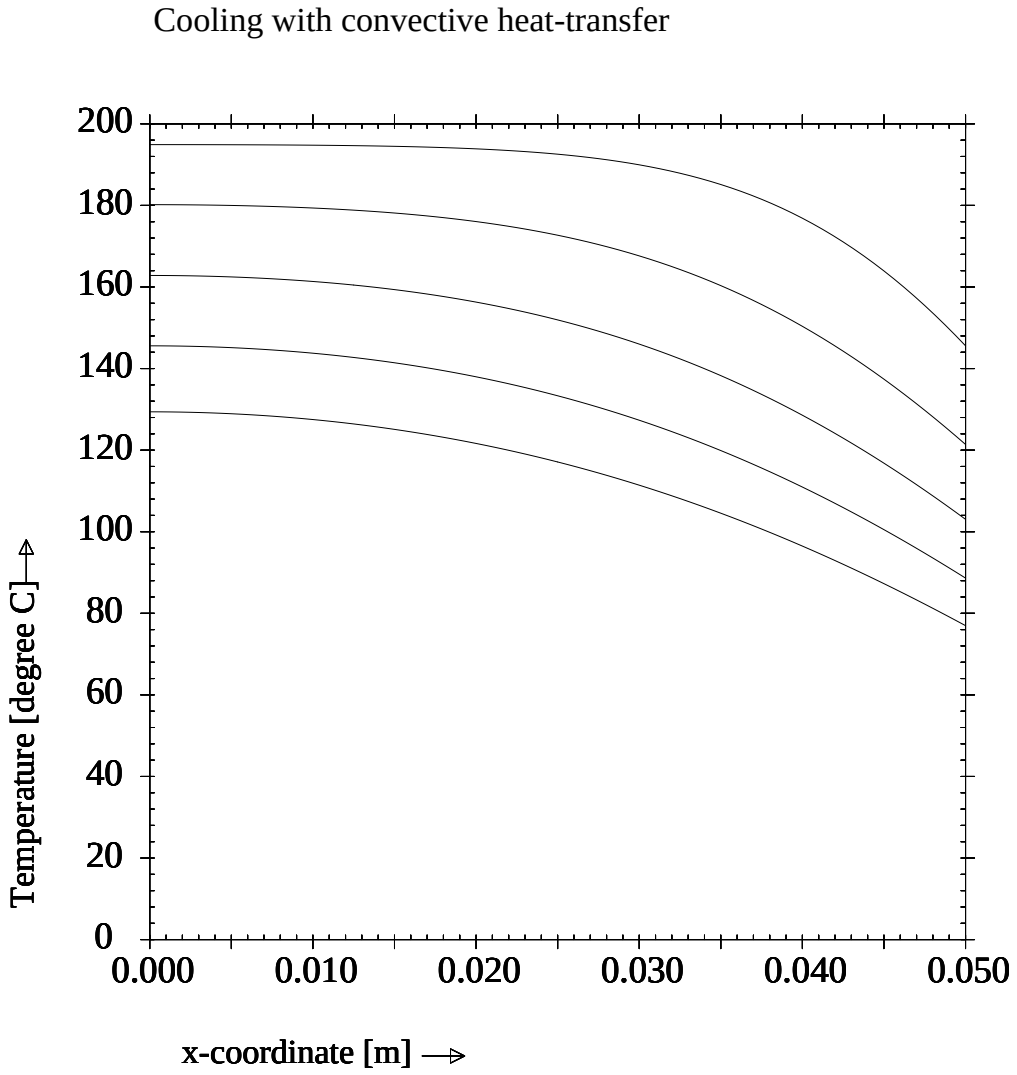


Figure 3.1.5.1: Time history at line $y = 0$

Figure 3.1.5.2 shows the coloured temperature distribution at time 2000 sec.
Figure 3.1.5.3 shows the isotherms at time 2000 sec.

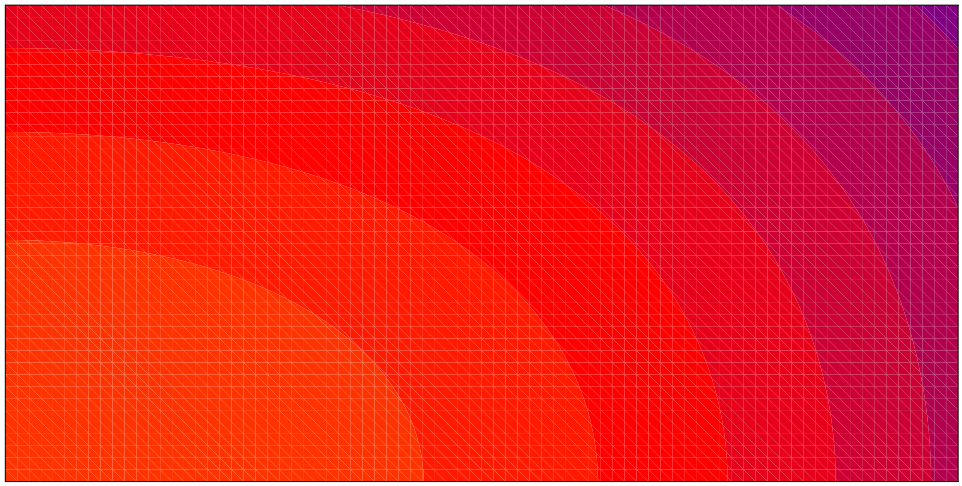


Figure 3.1.5.2: Temperature distribution at time 2000 seconds

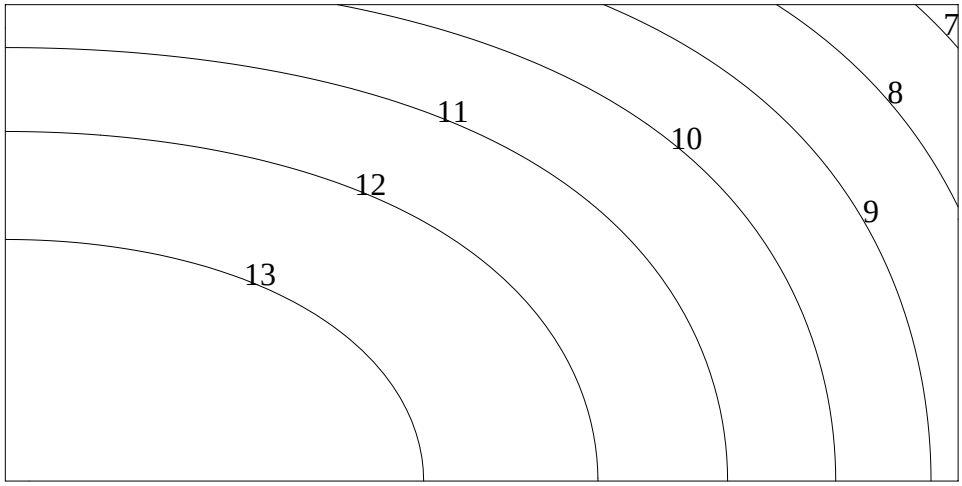


Figure 3.1.5.3: Isotherms at time 2000 seconds

3.1.6 Iterative solution of layered problems

In this section we shall focus ourselves on some aspects special for layered problems. With layered problems we mean problems with large contrasts in the coefficients.
A typical example of such a problem is the computation of excess pressures in the underground. Usually this concerns computations over a period of many millions of years and regions with a surface of the size of 20 to 50 km in both directions and a depth of several kilometers. In the underground we have layers that are relatively permeable, like sandstone layers and layers that are nearly impermeable (like shale or rock). The quotient of the permeabilities in such layers may be a factor of 10^7.
The result of such large contrasts in permeabilities is that the solution matrix becomes very ill-conditioned. The ill-conditioning is not so bad that the matrix becomes singular, in fact a direct solver does not have a problem solving the system of equations. However, for an iterative solver such a bad condition may lead to very large numbers of iterations and large computation times. Unfortunately for large three-dimensional problems direct solvers are much to slow and require too much memory. So actually it is necessary to solve such problems iteratively.
In this section we shall show how one can solve this problem by an iterative solver without having problems with the bad condition of the matrix. For a theoretical background the reader is referred to Vuik et al (1998).

For the sake of demonstration we consider only academic problems, which however, contain all difficulties present in this type of problems. First we consider a two-dimensional cross-section of part of the underground, consisting of 7 straight layers. The top layer consists of sandstone, the second one is shale, followed by a sandstone layer and so on. The region is sketched in Figure 3.1.6.1. In this region we solve the linearized 2D diffusion equation

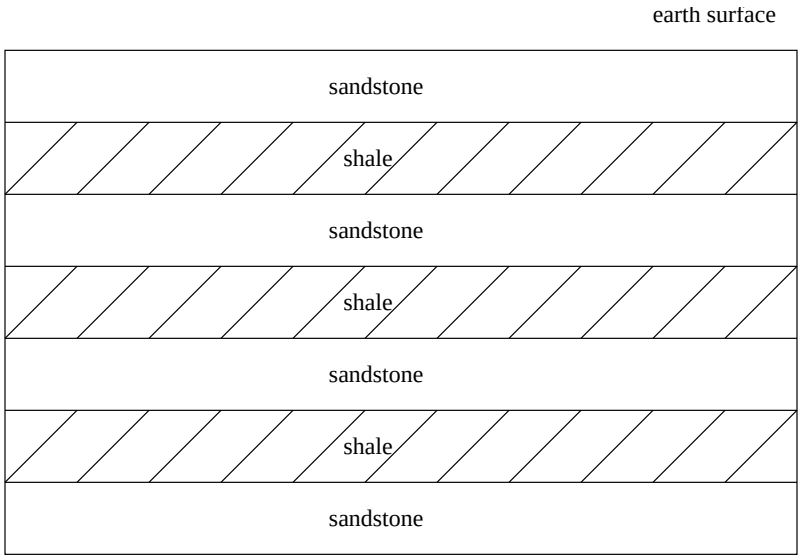


Figure 3.1.6.1: Artificial configuration with 7 straight layers

$$-\mathrm{div}(\sigma \nabla p) = 0\ , \tag{3.1.6.1}$$

with p the excess pressure and σ the permeability. At the earth's surface the excess pressure is prescribed.

For our model problem we assume that σ in sandstone is equal to 1 and σ in shale is equal to 10^{-7}. Furthermore the Dirichlet boundary condition at the earth's surface is set equal to 1. The solution of equation (3.1.6.1) with these boundary conditions is of course $p = 1$, but if we start with $p = 0$ or a random vector, our linear solver will not notice the difference with a real problem. Numerical

experiments show that the choice of one of these start vectors has only marginal effects.

In first instance we solve this equation by a Conjugate Gradient solver, without preconditioner. After that we consider the effect of an ILU preconditioning and finally we study the behaviour of the projection method mentioned in the Users Manual Section 3.2.8.
After the straight layers problem we consider the case of a curved mesh, and finally the case in which the projection vectors computed in the straight layer mesh are reused for the curved mesh. To get these examples into your local directory use the command `sepgetex` as follows:

```
sepgetex layerstr01    (7 straight layers, no preconditioning)
sepgetex layerstr02    (7 straight layers, ILU preconditioning)
sepgetex layerstr03    (7 straight layers, ILU preconditioning, with projection)
sepgetex layerarc01    (7 curved layers, ILU preconditioning, with projection)
sepgetex layerarc02    (7 curved layers, ILU preconditioning, with projection,
                        projection vectors created by straight layer mesh)
```

To run these examples use

```
  sepmesh layerstr01.msh
  sepcomp layerstr01.prb
```

and so on for all examples. There are no postprocessing files since the solution itself is trivial.

The mesh input file for the straight-layer problem is given by:

```
#  layerstr01.msh
#
#  mesh file for straight layer problem
#  Test without preconditioning
#  See Manual Examples Section 3.1.6
#
#  To run this file use:
#     sepmesh layerstr01.msh
#
#  Creates the file meshoutput
#

set warn off   ! suppress warnings
set time off   ! suppress printing of time

#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      nelm1 = 10          # number of elements in horizontal direction
      nelm2 = 5           # number of elements in vertical direction
   reals
      width  = 1          # width of the region
      height = 7          # height of the region
      h1 = 1              # top of 1-th layer
      h2 = 2              # top of 2-th layer
      h3 = 3              # top of 3-th layer
      h4 = 4              # top of 4-th layer
      h5 = 5              # top of 5-th layer
      h6 = 6              # top of 6-th layer
      h7 = 7              # top of 7-th layer
```

```
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1 =(0,0)                 # point left under
      p2 =( width,0)            # point right under
      p3 =(0, h1)               # left top of 1-th layer
      p4 =( width, h1)          # right top of 1-th layer
      p5 =(0, h2)               # left top of 2-th layer
      p6 =( width, h2)          # right top of 2-th layer
      p7 =(0, h3)               # left top of 3-th layer
      p8 =( width, h3)          # right top of 3-th layer
      p9 =(0, h4)               # left top of 4-th layer
      p10=( width, h4)          # right top of 4-th layer
      p11=(0, h5)               # left top of 5-th layer
      p12=( width, h5)          # right top of 5-th layer
      p13=(0, h6)               # left top of 6-th layer
      p14=( width, h6)          # right top of 6-th layer
      p15=(0, h7)               # left top of 7-th layer
      p16=( width, h7)          # right top of 7-th layer
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1 =line1(p1,p2,nelm= nelm1)       # straight line at bottom
      c2 =line1(p1,p3,nelm= nelm2)       # left-hand side of 1-th layer
      c3 =translate c1(p3,p4)            # right-hand side of 1-th layer
      c4 =translate c2(p2,p4)            # upper side of 1-th layer
      c5 =line1(p3,p5,nelm= nelm2)       # left-hand side of 2-th layer
      c6 =translate c1(p5,p6)            # upper side of 2-th layer
      c7 =translate c5(p4,p6)            # right-hand side of 1-th layer
      c8 =line1(p5,p7,nelm= nelm2)       # left-hand side of 3-th layer
      c9 =translate c1(p7,p8)            # upper side of 3-th layer
      c10=translate c8(p6,p8)            # right-hand side of 3-th layer
      c11=line1(p7,p9,nelm= nelm2)       # left-hand side of 4-th layer
      c12=translate c1(p9,p10)           # upper side of 4-th layer
      c13=translate c11(p8,p10)          # right-hand side of 4-th layer
      c14=line1(p9,p11,nelm= nelm2)      # left-hand side of 5-th layer
      c15=translate c1(p11,p12)          # upper side of 5-th layer
      c16=translate c14(p10,p12)         # right-hand side of 5-th layer
      c17=line1(p11,p13,nelm= nelm2)     # left-hand side of 6-th layer
      c18=translate c1(p13,p14)          # upper side of 6-th layer
      c19=translate c17(p12,p14)         # right-hand side of 6-th layer
      c20=line1(p13,p15,nelm= nelm2)     # left-hand side of 7-th layer
      c21=translate c1(p15,p16)          # upper side of 7-th layer
      c22=translate c20(p14,p16)         # right-hand side of 7-th layer
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
```

```
      s1=rectangle3(c1,c4,-c3,-c2)       # 1-th layer
      s2=rectangle3(c3,c7,-c6,-c5)       # 2-th layer
      s3=rectangle3(c6,c10,-c9,-c8)      # 3-th layer
      s4=rectangle3(c9,c13,-c12,-c11)    # 4-th layer
      s5=rectangle3(c12,c16,-c15,-c14)   # 5-th layer
      s6=rectangle3(c15,c19,-c18,-c17)   # 6-th layer
      s7=rectangle3(c18,c22,-c21,-c20)   # 7-th layer
#
#  Connect surfaces with element groups and provide them with one integer
#  property
#  Integer property 1 = 1 means normal permeability (sandstone)
#  Integer property 1 = 0 means low permeability (shale)
#
   meshsurf         # See Users Manual Section 2.2
      selm1 = s1, int_property 1 = 1    # 1-th layer (sandstone)
      selm2 = s3, int_property 1 = 1    # 3-th layer (sandstone)
      selm3 = s5, int_property 1 = 1    # 5-th layer (sandstone)
      selm4 = s7, int_property 1 = 1    # 7-th layer (sandstone)
      selm5 = s2, int_property 1 = 0    # 2-th layer (shale)
      selm6 = s4, int_property 1 = 0    # 4-th layer (shale)
      selm7 = s6, int_property 1 = 0    # 6-th layer (shale)

   plot                            # make a plot of the mesh
                                   # See Users Manual Section 2.2
end
```

The corresponding problem input file is given by

```
# layerstr01.prb
#
#  problem file for the straight layer problem
#  Test without preconditioning
#  See Manual Examples Section 3.1.6
#
#  To run this file use:
#     sepcomp layerstr01.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#

set warn off   ! suppress warnings

#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      k_shale = 1e-7     # scaled permeability for shale
      k_sand  = 1        # scaled permeability for sandstone
   vector_names
      pressure
      exact_pressure
   variables
```

```
      error
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
      elgrp1 = 800              # type number for Laplacian type equations
      elgrp2 = 800
      elgrp3 = 800
      elgrp4 = 800
      elgrp5 = 800
      elgrp6 = 800
      elgrp7 = 800
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
      curves0(c21)         # The pressure on the upper surface is 1
end

#  Define the structure of the large matrix

matrix                       # See Users Manual Section 3.2.4
   storage_method = compact, symmetric   # Symmetric compact storage,
                                         # hence an iterative method is used
end

#  Fill the non-zero values of the essential boundary conditions
#  See Users Manual Section 3.2.5

essential boundary conditions
    curves(c21),value=1    # The pressure on the upper surface is 1
end

#  Define the coefficients for the problems
#  See Users Manual Section 3.2.6
#  See also standard problems Section 3.1
coefficients
   elgrp1(nparm=20)               # coefficients for second order equation
                                  # Layer 1 (sandstone)
      coef6  =  k_sand            # a11 = kappa
      coef9  = coef6              # a22 = a11
   elgrp2(nparm=20)
      coef6  =  k_sand            # Layer 3 (sandstone)
      coef9  = coef6              #
   elgrp3(nparm=20)
      coef6  =  k_sand            # Layer 5 (sandstone)
      coef9  = coef6              #
   elgrp4(nparm=20)
      coef6  =  k_sand            # Layer 7 (sandstone)
      coef9  = coef6              #
   elgrp5(nparm=20)
      coef6  =  k_shale           # Layer 2 (shale)
      coef9  = coef6              #
   elgrp6(nparm=20)
```

```
      coef6  =  k_shale           # Layer 4 (shale)
      coef9  = coef6              #
   elgrp7(nparm=20)
      coef6  =  k_shale           # Layer 6 (shale)
      coef9  = coef6              #
end

# Input for the linear solver
# See users manual, Section 3.2.8

solve
   iteration_method = cg, preconditioning = none, accuracy = 1e-8//
   print_level = 2, start=zero, max_iter = 10000
   iseq_exact=exact_pressure
end

#
#   Create vector with exact solution (p=1)
#
create vector  # See users manual, Section 3.2.10
   value = 1
end


# Define the steps that must be carried out by the main program and the
# sequence of these steps

structure          # See users manual, Section 3.2.3

   create_vector, exact_pressure
   prescribe_boundary_conditions, pressure
   solve_linear_system, pressure
   error =  norm_dif=3, vector1 = exact_pressure, vector2 = pressure
   print error, text = 'difference '
   output

end
```

Mark that in the solve input block we have required an accuracy of $10^(-8)$. This may seem overdone but will be clear after the explanation. Furthermore the option `iseq_exact=1` is used to compare the numerical solution with the true solution. In this way we can compute the true error in each iteration step. The option `max_iter = 10000` is just a large overestimate. SEPRAN reduces this value to 10 times the number of unknowns.
Figure 3.1.6.2 shows the norm of the residual, the norm of the error and also the estimate of the smallest eigenvalue as function of the number of iterations. In each layer 10 elements in the horizontal and 5 elements in the vertical direction are used. From this figure the following remarkable observations may be made.

1. The residual decreases monotonically between iterations 1 and 30. For the iterations between 31 and 1650 we have an erratic behaviour of the residual. After iterations 1650 again we have a monotone decreasing of the residual.

2. If we require an accuracy of order 10^{-2}, the process would stop after approximately 25 iterations, since then the residual divided by the estimate of the smallest eigenvalue is small enough. Unfortunately the true error ($\|x - x_k\|_2$) is still large. The estimated error is not sharp, because the estimate of the smallest eigenvalue is very inaccurate.

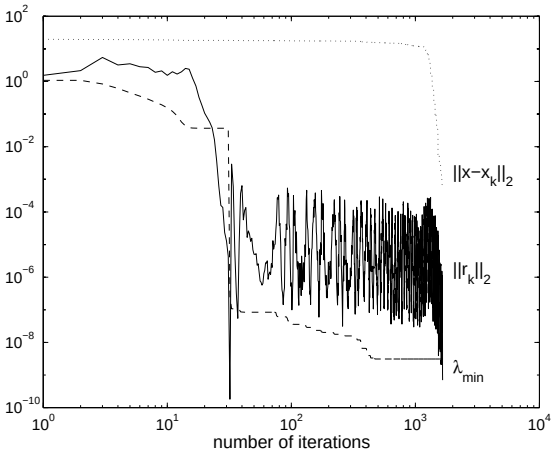


Figure 3.1.6.2: Convergence behaviour of CG without preconditioning

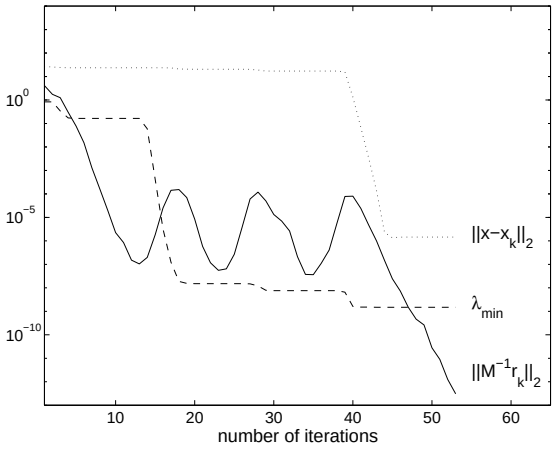


Figure 3.1.6.3: Convergence behaviour of CG with ILU preconditioning

3. In iterations 1-30 it looks as if the smallest eigenvalue is of order 10^{-2}, whereas from iteration 31 it is clear that the smallest eigenvalue is of order 10^{-7}.

So we see that the bad condition leads to a large number of iterations. Moreover, for practical values of the error, the termination criterion is not reliable.

Repeating the same experiment using an ILU preconditioning (also called ICCG) gives a drastic reduction of the number of iterations, but still the same conclusions as for the case without preconditioning can be drawn. Figure 3.1.6.3 shows the convergence behaviour. Note that the horizontal scales in Figures 3.1.6.2 and 3.1.6.3 are quite different. Although the number of iterations (48) is small compared to the non-preconditioned algorithm (1650), still it is quite large compared to the number of unknowns (385).
The mesh input file for the preconditioned case is identical to that of the non-preconditioned one. In the problem file only the `solve` input block is different

```
solve, sequence_number = 1
   iteration_method = cg, preconditioning = ilu, accuracy = 1e-8//
   print_level = 2, start=zero, max_iter = 10000//
   iseq_exact=exact_pressure
end
```

The graph of the residual in Figure 3.1.6.3, shows three bumps. This suggests that after the preconditioning there are three small eigenvalues in the spectrum of the preconditioned matrix. The reason why there are exactly three of such eigenvalues is explained in Vuik et al (1998). In order to accelerate the convergence and moreover to make the termination criterion reliable we try to approximate the corresponding eigenvectors and remove the contribution of these eigenvectors by a projection algorithm. This method is called the deflated ICCG method.
An important aspect is of course, how to approximate the eigenvectors. In order to solve this problem we solve Equation 3.1.6.1 for each of the shale layers separately with suitable boundary conditions. The solution of these problems is relatively easy, since σ is constant in a shale layer and the number of unknowns per layer is much smaller than in the original problem.
SEPRAN is only able to know how many small eigenvalues can be expected and how the approximate eigenvectors must be computed if it knows which layers correspond to a large permeability and which layers correspond to a small permeability. This can of course be verified by computing the value of σ in each element, but that process is time consuming and does not fit in the present way of dealing with the coefficients. To simplify the task it has been decided to provide each layer with exactly one integer property. In the input file `layerstr01.msh` it has been demonstrated how this is done.

Integer property 1 = 1 means a large permeability (sandstone) and integer property 1 = 0 means a low permeability (shale).
In order to activate the computation of the approximate eigenvectors and use of the projection method the `solve` input block is adapted as follows:

```
solve, sequence_number = 1
   iteration_method = cg, preconditioning = ilu, accuracy = 1e-8//
   print_level = 2, start=zero, max_iter = 10000//
   iseq_exact=exact_pressure, proj_method = approximate_eigenvectors//
   proj_accuracy=1d-2, proj_ignore = 1d-3
end
```

New in this case are the keywords `proj_method`, `proj_accuracy` and `proj_accuracy`.
`proj_method = approximate_eigenvectors` indicates that the projection method with approximate eigenvectors is used and since no keyword `proj_keep` is given, the eigenvectors are computed in this program.
`proj_accuracy=1d-2` defines how accurate the subproblem on the shale layer must be solved. An accuracy of $10^(-2)$ is sufficient in most practical applications.
Finally `proj_ignore = 1d-3` indicates that all elements in the projection vector that are smaller than $10^(-3)$ will be neglected. This may save computing time and memory, although for this particular problem there is no need to use it.
Numerical experiments have shown that the deflated ICCG method is approximately 30% more expensive per iteration. But since the number of iterations reduces considerably and moreover the termination criterion becomes reliable, this approach is a clear improvement compared to the classical ICCG method. Figures 3.1.6.4 and 3.1.6.5 show the convergence behaviour of the deflated method (noted as DICCG2) and the norm of the error for the ICCG and DICCG2 method.

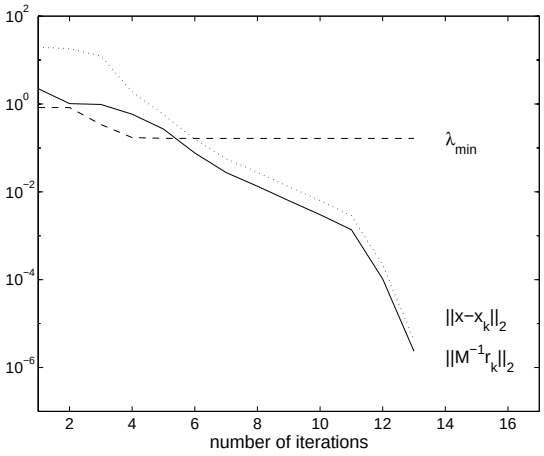


Figure 3.1.6.4: Convergence behaviour of DICCG2 for the straight layers problem

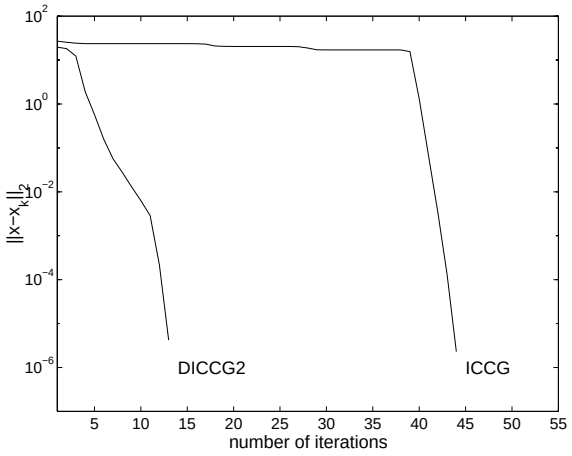


Figure 3.1.6.5: Norm of the error for the straight layers problem

Now we have seen that the deflated ICCG method (the projection method) behaves well for the straight layer problem we also apply it to an artificial curved example.
To the end the following mesh input file is used:

```
#  layerarc01.msh
#
#  mesh file for curved layer problem
#  See Manual Examples Section 3.1.6
#
#  To run this file use:
#     sepmesh layerarc01.msh
#
#  Creates the file meshoutput

set warn off   ! suppress warnings
set time off   ! suppress printing of time

#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      nelm1 = 10          # number of elements in horizontal direction
      nelm2 = 5           # number of elements in vertical direction
   reals
      width  = 1          # width of the region
      height = 7          # height of the region
      h1 = 1              # top of 1-th layer
      h2 = 2              # top of 2-th layer
      h3 = 3              # top of 3-th layer
      h4 = 4              # top of 4-th layer
      h5 = 5              # top of 5-th layer
      h6 = 6              # top of 6-th layer
      h7 = 7              # top of 7-th layer
      hw = 0.5            # centre of circle defining bottom line
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1 =(0,0)                 # point left under
      p2 =( width,0)            # point right under
      p3 =(0, h1)               # left top of 1-th layer
      p4 =( width, h1)          # right top of 1-th layer
      p5 =(0, h2)               # left top of 2-th layer
      p6 =( width, h2)          # right top of 2-th layer
      p7 =(0, h3)               # left top of 3-th layer
      p8 =( width, h3)          # right top of 3-th layer
      p9 =(0, h4)               # left top of 4-th layer
      p10=( width, h4)          # right top of 4-th layer
      p11=(0, h5)               # left top of 5-th layer
```

```
      p12=( width, h5)          # right top of 5-th layer
      p13=(0, h6)               # left top of 6-th layer
      p14=( width, h6)          # right top of 6-th layer
      p15=(0, h7)               # left top of 7-th layer
      p16=( width, h7)          # right top of 7-th layer
      p40 = ( hw,0)             # mid point of bottom line
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1 =arc1(p1,p2,-p40,nelm= nelm1)   # arc at bottom
      c2 =line1(p1,p3,nelm= nelm2)       # left-hand side of 1-th layer
      c3 =translate c1(p3,p4)            # right-hand side of 1-th layer
      c4 =translate c2(p2,p4)            # upper side of 1-th layer
      c5 =line1(p3,p5,nelm= nelm2)       # left-hand side of 2-th layer
      c6 =translate c1(p5,p6)            # upper side of 2-th layer
      c7 =translate c5(p4,p6)            # right-hand side of 1-th layer
      c8 =line1(p5,p7,nelm= nelm2)       # left-hand side of 3-th layer
      c9 =translate c1(p7,p8)            # upper side of 3-th layer
      c10=translate c8(p6,p8)            # right-hand side of 3-th layer
      c11=line1(p7,p9,nelm= nelm2)       # left-hand side of 4-th layer
      c12=translate c1(p9,p10)           # upper side of 4-th layer
      c13=translate c11(p8,p10)          # right-hand side of 4-th layer
      c14=line1(p9,p11,nelm= nelm2)      # left-hand side of 5-th layer
      c15=translate c1(p11,p12)          # upper side of 5-th layer
      c16=translate c14(p10,p12)         # right-hand side of 5-th layer
      c17=line1(p11,p13,nelm= nelm2)     # left-hand side of 6-th layer
      c18=translate c1(p13,p14)          # upper side of 6-th layer
      c19=translate c17(p12,p14)         # right-hand side of 6-th layer
      c20=line1(p13,p15,nelm= nelm2)     # left-hand side of 7-th layer
      c21=translate c1(p15,p16)          # upper side of 7-th layer
      c22=translate c20(p14,p16)         # right-hand side of 7-th layer
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle3(c1,c4,-c3,-c2)       # 1-th layer
      s2=rectangle3(c3,c7,-c6,-c5)       # 2-th layer
      s3=rectangle3(c6,c10,-c9,-c8)      # 3-th layer
      s4=rectangle3(c9,c13,-c12,-c11)    # 4-th layer
      s5=rectangle3(c12,c16,-c15,-c14)   # 5-th layer
      s6=rectangle3(c15,c19,-c18,-c17)   # 6-th layer
      s7=rectangle3(c18,c22,-c21,-c20)   # 7-th layer
#
#  Connect surfaces with element groups and provide them with one integer
#  property
#  Integer property 1 = 1 means normal permeability (sandstone)
#  Integer property 1 = 0 means low permeability (shale)
#
   meshsurf         # See Users Manual Section 2.2
      selm1 = s1, int_property 1 = 1    # 1-th layer (sandstone)
      selm2 = s3, int_property 1 = 1    # 3-th layer (sandstone)
      selm3 = s5, int_property 1 = 1    # 5-th layer (sandstone)
      selm4 = s7, int_property 1 = 1    # 7-th layer (sandstone)
      selm5 = s2, int_property 1 = 0    # 2-th layer (shale)
```

```
      selm6 = s4, int_property 1 = 0    # 4-th layer (shale)
      selm7 = s6, int_property 1 = 0    # 6-th layer (shale)

   plot                              # make a plot of the mesh
                                     # See Users Manual Section 2.2
end
```

The mesh is plotted in Figure 3.1.6.6 Numerical results for this mesh are comparable to the straight

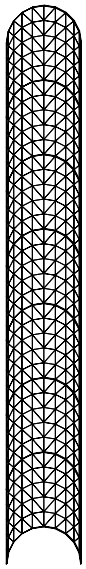


Figure 3.1.6.6: Mesh used in the parallel arcs layered problem

layer problem and will not be repeated here.

Finally we shall show how the method behaves if approximate eigenvectors computed in one configuration are used in the other one. To that end we start with a mesh consisting of straight layers and compute the approximate eigenvectors. After that we transform the coordinates such that the curved mesh arises. Instead of recomputing the approximate eigenvectors we reuse the eigenvectors computed in the straight layer case. Although these new eigenvectors are not as accurate as the ones directly computed on the curved mesh, the results are almost comparable. The space spanned by the eigenvectors of the straight layer problem does not differ too much of the space spanned by the eigenvectors of the curved mesh. So it is not necessary to know the approximate eigenvectors to accurately, as long as the main behaviour of the eigenvectors is present.
If we create the mesh for the straight layer problem and compare it with the curved mesh then we see an essential difference. In the straight layer problem all rectangles are subdivided into triangles that all are directed in the same direction. In the curved case, however the triangles at the left-hand side of the symmetry axis are directed in the opposite direction of that of the right-hand side. This is because the mesh generator tries to avoid large angles. If we start with the straight layer mesh and change the coordinates without precautions, all diagonals would be pointing in one direction. The results is an error message that the ILU preconditioning does not exist. This is due to the fact that the matrix is not longer diagonal dominant due to the large angles. This is typical for this extreme case.

In order to create diagonals pointing in the right direction we start with a curved mesh, where the centre of the arc defining the bottom line is defined by

```
p40 = (hw,-1000)          # centroid of bottom line
                          # In order to get a "straight line" the
                          # point is moved a large distance downwards
```

The rest of the input file is not changed. The result is an almost straight mesh since the radius of the circles is approximately equal to 1000, but the diagonals of the triangles are pointed in the right direction.

To change the coordinates of the mesh we use the option `change_coordinates` in the input block defined by the keyword `structure`. This requires an extra input block and also a function subroutine FUNCCOOR that defines the transformation from old to new coordinates. For that reason it is necessary to supply a new main program `layerarc02.f` with the following contents:

```
      program layerarc02

!     --- Main program for straight/curved layer problem
!         See Manual Examples Section 3.1.6
!         To link this program use:
!
!         seplink layerarc02

      call sepcom ( 0 )
      end

      subroutine funccoor ( ichoice, ndim, coor, nodes, numnodes )

!     --- This subroutine is used to change the coordinates
!         The input coordinates are for the straight mesh
!         The output coordinates are for the curved mesh
!         The transformation is given by:
!
!         x_curved = (1-cos(pi x_straight))/2
!         y_curved = y_straight+sin(pi x_straight)/2

      implicit none
      integer          ichoice, ndim, numnodes , nodes(numnodes)
      integer          i, nodenr
      double precision coor(ndim,*)
      include 'SPcommon/consta'
      do i = 1, numnodes
         nodenr = nodes(i)
         coor(2,nodenr) = coor(2,nodenr)+0.5d0*sin(pi*coor(1,nodenr))
         coor(1,nodenr) = 0.5d0*(1d0-cos(pi*coor(1,nodenr)))
      end do
      end
```

To link this program use the command seplink:

```
seplink layerarc02
```

The corresponding input file is almost identical to the file `layerstr03.prb`, except for the following parts:

```
# Input for the linear solver
# See users manual, Section 3.2.8

solve, sequence_number = 1
   iteration_method = cg, preconditioning = ilu, accuracy = 1e-8//
   print_level = 2, start=zero, max_iter = 10000//
   iseq_exact=exact_pressure, proj_method = approximate_eigenvectors//
   proj_accuracy=1d-2, proj_ignore = 1d-3, proj_keep = keep
end

solve, sequence_number = 2
   iteration_method = cg, preconditioning = ilu, accuracy = 1e-8//
   print_level = 2, start=zero, max_iter = 10000//
   iseq_exact=exact_pressure, proj_method = approximate_eigenvectors//
   proj_keep = old
end

#   To transform the coordinates from the straight mesh to the curved mesh
#   change_coordinates is used
#
#   See users manual, Sections 3.2.3 and 2.2
#
change_coordinates, sequence_number = 1
   all
end

# Define the steps that must be carried out by the main program and the
# sequence of these steps
# Vector 1 contains the exact solution
# Vector 2 contains the numerical solution of the straight mesh and later on
# of the curved mesh

structure          # See users manual, Section 3.2.3

   create_vector, exact_pressure
   prescribe_boundary_conditions, pressure
   solve_linear_system, pressure
   error =  norm_dif=3, vector1 = exact_pressure, vector2 = pressure
   print error, text = 'difference '
   output

end
```

First the exact solution is created, then the problem is solved on the straight layer mesh and the approximate eigenvectors are kept.
This is the option `proj_keep = keep`.
Next the coordinates are changed and the problem is solved again starting with the zero vector. The previously computed projection vectors are reused. This is the option `proj_keep = old`.

3.1.7 Stability of a salt layer formed by salty ground-water upflow

3.1.7.1 Outline of the problem

This problem has been studied by Gert-Jan Pieters (2000), for more mathematical background on the problem we refer to his Master's Thesis .

Upflowing salty ground-water in the subsurface evaporates completely at the surface. After through-flow induced by evaporation, the salt remains behind at the surface (salt-lakes). This saline layer is referred to as a diffusion layer which may grow up to a finite thickness. This finite thickness is an equilibrium between upflowing salt in solution and downward diffusion. It is our aim to analyze this natural process numerically.

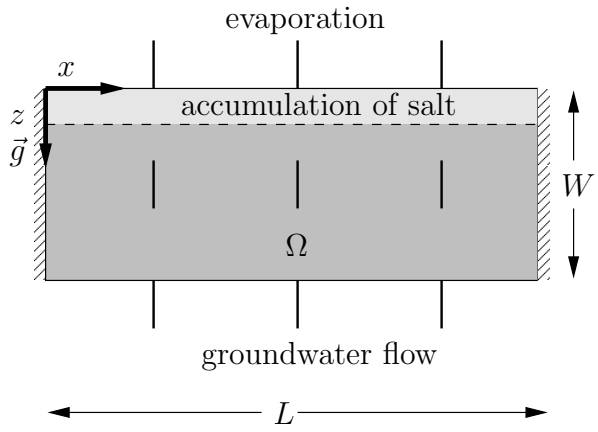


Figure 3.1.7.1: Geometry

Consider a bounded porous medium with a horizontal upper boundary (surface), see Figure 3.1.7.1. For the case of a uniform upflow within the medium and through the boundary, we treat the problem as one with one spatial dimension. However, for the case of a porous medium with non-homogeneous and non-isotropic permeability the problem has to be treated as a two-dimensional problem. Van Duijn et al (2000), Wooding (1960) found instabilities of the diffusion layer. These instabilities were triggered by perturbation of either the initial condition (locally or globally) or by local perturbation at all times. In this research the initial condition is globally perturbed (in this context globally means the interior of the domain Ω, or $\Omega/\partial\Omega$, see Figure 3.1.7.1). Wooding (1960) found instability of the diffusion layer numerically and his observation were confirmed by experiments. Van Duijn et al. (2000) analyzed these instabilities using semi-explicit expressions for an unbounded domain. In the present work we are concerned with analysis of the stability of this diffusion layer with respect to small perturbations of the initial condition of the saturation in the domain.

3.1.7.2 Equations for salt transport

We use the same equations as Van Duijn et al. (2000) and consider an isotropic, homogeneous medium. Let the water density, fluid density far away from the surface, local fluid density and maximum fluid density at the outflow boundary be denoted by ρ_0, ρ, ρ_r and ρ_m [kg/m^3] respectively. Clearly $\rho_0 < \rho_r < \rho_m$ and $\rho_r \leq \rho \leq \rho_m$.
Assuming the porosity ϕ [-] to be constant, we have for the fluid mass-balance equation:

$$\phi \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{q}) = 0, \tag{3.1.7.1}$$

where $\mathbf{q}$ [$m^3/(m^2 s)$] is the Darcy volume flow rate and t is time. We use bold-face characters to indicate that quantities are vectors. For the mass-balance of salt one obtains

$$\phi \frac{\partial(\rho\omega)}{\partial t} + \nabla \cdot (\rho\omega\mathbf{q} - \rho\mathbf{D}\nabla\omega) = 0, \tag{3.1.7.2}$$

where ω is the mass fraction of salt (i.e. salt mass per unit fluid volume). The dispersivity is given by $\mathbf{D}$. The equation of state is taken as (see van Duijn et al (1993))

$$\rho = \rho_0 e^{\alpha\omega}, \tag{3.1.7.3}$$

where α is treated as a constant. The volume flow rate follows from Darcy's Law:

$$\frac{\mu}{\kappa}\mathbf{q} + \nabla(p - g\rho_r z) - (\rho - \rho_r)g\mathbf{k} = \mathbf{0}. \tag{3.1.7.4}$$

Here p, g, κ, μ are pressure, gravity constant, permeability and fluid viscosity respectively. Combination of equations 3.1.7.1, 3.1.7.2 and 3.1.7.3 gives

$$\phi \frac{\partial\rho}{\partial t} + \mathbf{q} \cdot \nabla\rho = \mathbf{D}\Delta\rho. \tag{3.1.7.5}$$

Here Δ denotes the Laplacian operator. In order to simplify the subsequent analysis, we apply the Boussinesq approximation. The approximation consists of setting constant all the properties of the medium, except that the buoyancy term is retained in the Darcy equation. As a consequence the equation of continuity reduces to $\div\mathbf{q} = 0$. The Boussinesq is valid provided that density changes remain small in comparison to ρ_r.

3.1.7.3 Dimensionless equations

Introduce the saturation

$$S := \frac{\rho - \rho_r}{\rho_m - \rho_r}, \quad \text{with } 0 \le S \le 1, \tag{3.1.7.6}$$

and define the dimensionless vector $\mathbf{U}$ proportional to volume flow rate:

$$\mathbf{U} := \frac{\mu\mathbf{q}}{(\rho_m - \rho_r)g\kappa}, \tag{3.1.7.7}$$

subsequently we replace t by a dimensionless time $\tau := \dfrac{t\varepsilon^2}{\phi\mathbf{D}}$, where ε is the rate of through-flow by evaporation through the surface. The Cartesian coordinates (x, y, z) are scaled to the thickness of the equilibrium boundary layer, $\delta = \mathbf{D}/\varepsilon$. Finally we introduce the scale for the pressure p as

$$P := \frac{p - \rho_r g\delta z}{(\rho_m - \rho_r)g\delta}. \tag{3.1.7.8}$$

The dimensionless equations become

$$(P_1) \begin{cases} \nabla \cdot \mathbf{U} = 0, \\ \mathbf{U} + \nabla P - S\mathbf{k} = \mathbf{0}, \\ \frac{\partial S}{\partial \tau} + \mathrm{Ra}\mathbf{U} \cdot \nabla S = \Delta S. \end{cases}$$

Here the Rayleigh-number Ra is defined as $\mathrm{Ra} := \dfrac{(\rho_m - \rho_r)g\kappa}{\mu\varepsilon}$. It is our aim to analyze stability of the diffusion layer with respect to small perturbations of the initial saturation profile for different Rayleigh numbers. Problem (P_1) has boundary conditions and initial conditions

$$(IB_1) \begin{cases} S(x, z, 0) = 0, & (x, y) \in \Omega, \\ S(x, 0, \tau) = 1, & \tau > 0,\ 0 \le x \le L, \\ S(x, W, \tau) = 0, & \tau > 0,\ 0 \le x \le L, \\ \left.\dfrac{\partial S}{\partial x}\right|_{x=0,L} = 0, & \tau > 0,\ 0 \le z \le W, \\ \mathbf{U}|_{z=0,W} = -\varepsilon\mathbf{k}, & \tau > 0,\ 0 \le x \le L, \\ \left.\dfrac{\partial P}{\partial x}\right|_{x=0,L} = 0, & \tau > 0,\ 0 \le z \le W. \end{cases}$$

Since above conditions and P_1 imply that the pressure is determined up to an integration constant, we use the numerically superior stream-function formulation to solve the two-dimensional problem.

3.1.7.4 The stream function formulation

We introduce a "vector potential" such that $\mathbf{U} = \operatorname{curl} \mathbf{\Psi}$, which reassures that $\div \mathbf{U} = 0$, since we always have $\div(\operatorname{curl} \mathbf{\Psi}) = 0$. Furthermore, since $\mathbf{U} = (U_x, 0, U_z)$ and all differentiations with respect to y vanish, i.e. $\partial_y = 0$, we have $\mathbf{\Psi} = (0, \Psi_y, 0)$. We substitute $\mathbf{U} = \operatorname{curl} \mathbf{\Psi}$ into P_1 and take the curl of the third equation of P_1 and keep in mind that $\operatorname{curl}(\operatorname{grad} P) = \mathbf{0}$. Hirasaki & Hellums (1968) prove that a vector potential $\mathbf{\Psi}$ exists and is solenoidal if the velocity field $\mathbf{U}$ is solenoidal, i.e. $\div \mathbf{\Psi} = 0$. Keeping this in mind we obtain

$$(P_2) \begin{cases} \dfrac{\partial S}{\partial \tau} + \mathrm{Ra}\left(-\dfrac{\partial \Psi_y}{\partial z}\dfrac{\partial S}{\partial x} + \dfrac{\partial \Psi_y}{\partial x}\dfrac{\partial S}{\partial z} \right) = \Delta S, \\ \Delta \Psi_y = \dfrac{\partial S}{\partial x}. \end{cases}$$

Here Δ denotes the Laplacian in the x and z co-ordinates. The initial and boundary conditions change into

$$(IB_2) \begin{cases} S(x,z,0) = 0, & (x,y) \in \Omega, \\ S(x,0,\tau) = 1, & \tau > 0,\ 0 \le x \le L, \\ S(x,W,\tau) = 0, & \tau > 0,\ 0 \le x \le L, \\ \left.\dfrac{\partial S}{\partial x}\right|_{x=0,L} = 0, & \tau > 0,\ 0 \le z \le W, \\ \Psi_y\big|_{z=0,W} = -\varepsilon x, & \tau > 0,\ 0 \le x \le L, \\ \Psi_y(0,z,\tau) = 0, & \tau > 0,\ 0 \le z \le W, \\ \Psi_y(L,z,\tau) = -\varepsilon L, & \tau > 0,\ 0 \le z \le W. \end{cases}$$

3.1.7.5 Stability

We analyze P_2 and IB_2 with respect to small perturbations of the initial saturations, i.e.

$$\widetilde{S} = S + \epsilon \nu, \tag{3.1.7.9}$$

where S comes from (P_2) and (IB_2), $\widetilde{S}$ is the perturbed saturation and $\nu = \nu(x)$ is the perturbation function. The magnitude of the perturbation is given by ϵ.
We are interested in the behaviour of the L_2-norm of the gradient of the perturbed stream-function, i.e. $\int\limits_\Omega |\nabla(\Psi_y - \widetilde{\Psi}_y)|^2$, where $\widetilde{\Psi}_y$ is the perturbed stream-function. We denote this integral as $||\nabla(\Psi_y - \widetilde{\Psi}_y)||_{L_2(\Omega)}$. Since the unperturbed problem is one-dimensional, we have

$$\begin{cases} \Delta \Psi_y = 0, & \tau \ge 0 \quad \text{(unperturbed)}, \\ \Delta \widetilde{\Psi}_y = \epsilon \frac{\partial \nu}{\partial x}, & \tau = 0 \quad \text{(perturbed)}. \end{cases}$$

In the stable case it can be shown that $||\nabla(\Psi_y - \widetilde{\Psi}_y)||_{L_2(\Omega)} \le \epsilon^2 ||\nu||_{L_2(\Omega)}$ for all $\tau \ge 0$. We qualify the system stable for perturbations when

$$\frac{d}{d\tau}||\nabla(\Psi_y - \widetilde{\Psi}_y)||_{L_2(\Omega)} < 0. \tag{3.1.7.10}$$

It turns out that often $\dfrac{d}{dt}|||\nabla(\Psi_y - \widetilde{\Psi}_y)|||_{L_2(\Omega)} < 0$ for some time $0 < \tau < \tau^*$ and $\dfrac{d}{d\tau}|||\nabla(\Psi_y - \widetilde{\Psi}_y)|||_{L_2(\Omega)} > 0$ when $\tau > \tau^*$. The routines developed in this problem keeps track of the L_2-norm of the gradient of the perturbed stream function, i.e. $|||\nabla(\Psi_y - \widetilde{\Psi}_y)|||_{L_2(\Omega)}$, and gives output in terms of the stream-function, saturation and velocities.

3.1.7.6 Examples

We show an example of a perturbation $\nu = \sin ax$, $a = 0.25$, $\epsilon = 0.001$ and geometrical settings $L = 50$, $W = 5$ and Ra $= 5$. Furthermore, we show the evaluation of $|||\nabla(\Psi_y - \widetilde{\Psi}_y)|||_{L_2(\Omega)}$ as function of time. We see that this norm decreases monotonically and hence the small fluctuations are damped. See Figures 3.1.7.2, 3.1.7.3, 3.1.7.4 and 3.1.7.5. As a counter example we show a calculation with the same settings, except Ra $= 35$. Now we see that S contains fingers and Ψ_y and $\mathbf{U}$ give rotations. The norm $|||\nabla(\Psi_y - \widetilde{\Psi}_y)|||_{L_2(\Omega)}$ decreases for some time and increases subsequently, indicating its unstable behaviour with respect to small initial perturbations. See Figures 3.1.7.6, 3.1.7.7, 3.1.7.8 and 3.1.7.9.

Future numerical analysis for this problem:

- further analysis of the instabilities
- different initial perturbation functions, e.g. random perturbations
- non-homogeneous and anisotropic media

3.1.7.7 SEPRAN files

To get the files into your local directory use

```
sepgetex salt_stable
```

The mesh, problem, postprocessing and Fortran code files are given below.
The mesh input file

```
* salt_stable.msh
*
* mesh for natural convection problem
*
constants
   integers
      nx  = 200
      nz  = 20
   reals
      length = 50
      depth  = 5
end
mesh2d
   points
      p1=(0,0)
      p2=( length,0)
      p3=( length, depth)
      p4=(0, depth)
   curves
      c1=line 1(p1,p2, nelm= nx,ratio=1, factor=1)
      c2=line 1(p2,p3, nelm= nz,ratio=1, factor=1)
      c3=line 1(p3,p4, nelm= nx,ratio=1, factor=1)
      c4=line 1(p4,p1, nelm= nz,ratio=1, factor=1)
   surfaces
      s1=rectangle5(c1,c2,c3,c4)
   meshsurf
      selm1 = s1
   plot (jmark=5, numsub=1)
end
```

The main program and related subroutines:

```
! salt_stable.f
!
! This file contains additional subroutines
!
      program salt_stable
      implicit none
      call sepcom ( 0 )
      end

! *********************************************
!
!    functions for essential boundary conditions
!
! *********************************************
      function funcbc ( ichoice, x, y, z )
      implicit none
      double precision funcbc, x, y, z, R, getconst
      integer ichoice
      include 'SPcommon/ctimen'

      R = getconst('R')

      if ( ichoice==1 ) then
         funcbc = -1 * (1/R) * (x)
      end if

      if ( ichoice==2) then
         funcbc = -1 * (1/R) * (x)
      end if

      end

! *********************************************
!
!    perturbation of the initial saturation
!
! *********************************************
      function func ( ichoice, x, y, z )
      implicit none
      double precision func, x, y, z, R, Depth, Length, a, getconst
      integer ichoice
      include 'SPcommon/ctimen'

      R = getconst('R')
      Depth = getconst('Depth')
      Length = getconst('Length')
      a = getconst('a')

      if (ichoice==1) then
         if ( (y.lt.Depth).and.(y.gt.0)
     +           .and.(x.gt.0).and.(x.lt.Length) ) then
           func = 0.001 * sin(a * x)
         else
```

```
          func = 0
         end if
      end if

      end


      subroutine compcons
      implicit none
      double precision R, R_inv, Length_div_R, Length, getconst

      R = getconst('R')
      Length = getconst('Length')

      R_inv = 1/R
      Length_div_R = Length/R

      call putreal ( 'R_inv', R_inv )
      call putreal ( 'min_R_inv', -R_inv )
      call putreal ( 'Length_div_R', Length_div_R )
      call putreal ( 'min_Length_div_R', -Length_div_R )

      end
```

The input file for the computational program:

```
* salt_stable.prb
*
* For details, see the text above
*

constants
   reals
      R = 35
      R_inv
      min_R_inv
      Length_div_R
      min_Length_div_R
      D = 1
      Depth = 5
      Length = 50
      a = 0.25

   vector_names
     min_R_inv
     delta_grad_Psi
     q
     S
     dS_dx
     min_dS_dx
     Psi
     Psi_x
     Psi_z
     R_Psi_x
     R_Psi_z
```

```
     min_R_Psi_x
     min_R_Psi_z
     Psi_x_min_R_inv
     norm_grad_Psi

   variables
      res_int
end

problem 1 # stream-function equation
   types
      elgrp1 = (type = 800)
  essboundcond
      curves(c1)
      curves(c2)
      curves(c3)
      curves(c4)

problem 2 # saturation equation
   types
      elgrp1 = (type = 800)
   essboundcond
      curves(c1)
      curves(c3)
end

*
* Computations structure
*
structure

   # create vector -1/R
   create_vector, sequence_number=2, min_R_inv

   # create delta_grad_Psi (initial with ones at both degrees of freedom)
   create_vector, sequence_number=6, delta_grad_Psi

   # create q (initial with ones at both degrees of freedom)
   create_vector, sequence_number=3, q

   # create perturbed startvector S
   create_vector, sequence_number=1, S

   # prescribe Dirichlet conditions for the saturation S
   prescribe_boundary_conditions, sequence_number=2, S

   # compute dS/dx
   derivatives, seq_deriv=3, dS_dx

   # compute min_dS_dx
   min_dS_dx = - dS_dx

   # prescribe boundary conditions for Psi
   prescribe_boundary_conditions, sequence_number=1, Psi
```

```
# solve pressure Psi
solve_linear_system, seq_coef=1, problem=1, Psi

# compute Psi_x
derivatives, seq_deriv=1, Psi_x

# compute Psi_z
derivatives, seq_deriv=2, Psi_z

# compute R times Psi_x
R_Psi_x = R * Psi_x

# compute min_R_Psi_x
min_R_Psi_x = - R_Psi_x

# compute R times Psi_z
R_Psi_z = R * Psi_z

# compute min_R_Psi_z
min_R_Psi_z = - R_Psi_z

# compute velocity q
copy min_R_Psi_z q degfd2=1
copy R_Psi_x q degfd2=2

# compute Psi_x - 1/R
Psi_x_min_R_inv = Psi_x - min_R_inv

# compute delta_grad_Psi
copy Psi_x_min_R_inv delta_grad_Psi degfd2=1
copy Psi_z delta_grad_Psi degfd2=2

# compute norm_grad_Psi
norm_grad_Psi = inner_product delta_grad_Psi delta_grad_Psi

# compute L2 norm of the velocity difference
integral, seq_coef=3, seq_integral=1, res_int, norm_grad_Psi
print res_int, text=' '

# write the solutions for t=0 to a file
output

# start first time loop
start_time_loop

   # compute time step
   time_integration, S

   # compute dS/dx
   derivatives, seq_deriv=3, dS_dx

   # compute min_dS_dx
   min_dS_dx = - dS_dx

   # prescribe the boundary conditions for Psi
```

```
      prescribe_boundary_conditions, sequence_number=1, Psi

      # solve pressure Psi
      solve_linear_system, seq_coef=1, problem=1, Psi

      # compute Psi_x
      derivatives, seq_deriv=1, Psi_x

      # compute Psi_z
      derivatives, seq_deriv=2, Psi_z

      # compute R times Psi_x
      R_Psi_x = R *Psi_x

      # compute min_R_Psi_x
      min_R_Psi_x = - R_Psi_x

      # compute R times Psi_z
      R_Psi_z = R *Psi_z

      # compute min_R_Psi_z
      min_R_Psi_z = - R_Psi_z

      # compute velocity q
      copy min_R_Psi_z q degfd2=1
      copy R_Psi_x q degfd2=2

      # compute Psi_x - 1/R
      Psi_x_min_R_inv = Psi_x - min_R_inv

      # compute delta_grad_Psi
      copy Psi_x_min_R_inv delta_grad_Psi degfd2=1
      copy Psi_z delta_grad_Psi degfd2=2

      # compute norm_grad_Psi
      norm_grad_Psi = inner_product delta_grad_Psi delta_grad_Psi

      # compute L2 norm of the velocity difference
      integral, seq_coef=3, seq_integral=1, res_int, norm_grad_Psi
      print res_int, text=' '

      # write solutions for each time step to a file
      output

   # end time loop
   end_time_loop

end

*
*  Define initial conditions for the saturation S
*
create vector, sequence_number=1, problem = 2
  func = 1
end
```

```
*
* Define min_R_inv
*
create vector, sequence_number = 2
  type = vector of special structure v1
  value =  min_R_inv
end

*
* Define q (initial)
*
create vector, sequence_number = 3
  type = vector of special structure v2
  value = 1, degfd = 1
  value = 1, degfd = 2
end

*
* Define delta_grad_Psi (initial)
*
create vector, sequence_number=6, problem=1
  type = vector of special structure v2
  value = 1, degfd = 1
  value = 1, degfd = 2
end

*
*  Essential boundary conditions for stream-function Psi
*
essential boundary conditions, sequence_number=1, problem=1
   curves (c1), func = 1
   curves (c4), value = 0
   curves (c3), func = 2
   curves (c2), value =  min_Length_div_R
end

*
*  Essential boundary conditions for saturation S
*
essential boundary conditions, sequence_number=2, problem=2
   curves (c3), value = 0
   curves (c1), value = 1
end

*
*   Derivatives block, to compute Psi_x
*
derivatives, sequence_number=1, problem=1
   icheld = 1, ix=1
   seq_input_vector =  Psi
end

*
*   Derivatives block, to compute Psi_z
```

```
*
derivatives, sequence_number=2, problem=1
   icheld = 1, ix=2
   seq_input_vector =  Psi
end

*
*   Derivatives block, to compute  dS/dx
*
derivatives, sequence_number=3, problem=2
   icheld = 1, ix=1
   seq_input_vector =  S
end

*
* Integral block, to compute the L_2_norm
*
integrals, sequence_number = 1
  icheli = 2
end

*
*  Definition of coefficients for the streamfunction
*
coefficients, sequence_number = 1, problem=1
   elgrp1(nparm=20)
     coef6 = 1
     coef9 = coef6
     coef16 = old_solution  min_dS_dx
end

*
*  Coefficients for the saturation equation
*
coefficients, sequence_number = 2, problem=2
   elgrp1(nparm=20)
      icoef2 = 1
      coef6 =  D
      coef9 = coef6
      coef17 = 1
      coef12 = old_solution  min_R_Psi_z
      coef13 = old_solution  R_Psi_x
end

*
*  Coefficients for the area integration
*
coefficients, sequence_number = 3, problem=1
  elgrp1(nparm=10)
    coef4 = 1
end

*
*  Definition of the time loop
*
```

```
time_integration
   method = euler_implicit
   tinit = 0
   tend  = 5
   tstep = 0.1
   toutinit = 0
   toutend  = 5
   toutstep = 0.1
   seq_boundary_conditions = 2
   seq_coefficients = 2
end
```

The seppost input file

```
* salt_stable.pst
*
*
* input for seppost
*
postprocessing
  time = (0,5)
     plot vector q
     plot contour Psi
     plot coloured contour S
  end
end
```

3.1.8 A comparison of some upwind schemes

In this section we consider a number of classical test schemes for upwind methods.
It concerns the following problems:

Convection skew to the mesh.

Rotating cone problem.

3.1.8.1 Convection skew to the mesh

In this example we consider a convection-diffusion problem, with zero source term. The diffusivity was taken to be 10^{-6}. The flow in the unit square is unidirectional and constant ($||\mathbf{u}|| = \mathbf{1}$). At the lower boundary of the square we have a Dirichlet boundary condition ($c = 1$). At the left-hand side we have also a Dirichlet boundary condition ($c = 1$ for $y \leq 0.2$ and $c = 0$ for $y > 0.2$). The angle α of the flow is an input parameter, which in our example is equal to 45°. At all other boundaries the natural boundary condition $\frac{\partial \phi}{\partial n} = 0$ is imposed.
Figure 3.1.8.1 shows the configuration used The result is a discontinuous concentration over the region. The exact solution is equal to 1 in the region starting with boundary condition 1 and following the straight line with the angle of the flow.
In this section we shall compare the behaviour of standard Galerkin and some upwind schemes for this problem. Both linear triangles and bi-linear quadrilaterals are used.
Quadratic elements do not behave so well for this kind of problems and it is advised always to use linear elements. If the velocity is the result of a quadratic velocity computation, the option `linear_subelements` will subdivide the quadratic elements into linear ones.
The exact solution satisfies $0 \leq c \leq 1$ and a scheme is said to satisfy the maximum principle if the numerical solution is also between 0 and 1.
If the result of a scheme does not satisfy the maximum principle we can always force this condition by using the keyword `limit_solution` either in the linear solver or the nonlinear solver. Of course this is brute force and also not accurate, but for some applications it is a must.

To get this example into your local directory use:

```
sepgetex conv_shockxx
```

with `xx` equal to 01 02 03 or 04.
These options correspond to the following cases:

01 Linear triangular elements

02 Linear triangular elements with limiting

03 Bi-linear quadrilaterals

04 Bi-linear quadrilaterals with limiting

To run these problems use:

```
sepmesh conv_shockxx.msh
sepview sepplot.001
seplink conv_shockxx
conv_shockxx < conv_shockxx.prb
seppost conv_shockxx.pst
sepview sepplot.001
```

The mesh input file for the linear triangle case is given by

```
#  conv_shock01.msh
#
#  mesh file for testing of upwind schemes for 2d convection-diffusion
#  linear triangular elements
#  See manual standard problems, Section 3.1.8.1
#  Shock problem
#
#  To run this file use:
#     sepmesh conv_shock01.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1               # width of the square
      length = 1              # length of the square
      discontinuity = 0.2     # height of the discontinuity point on left-hand
                              # side
   integers
      n  = 10          # number of elements in length direction
      m1 = 2           # number of elements in width direction from below to
                       # point with discontinuity
      m2 = 8           # number of elements in width direction from
                       # point with discontinuity to top
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)                      # Left under point
      p2=( length,0)                # Right under point
      p3=( length, width)           # Right upper point
      p4=(0, width)                 # Left upper point
      p5=(0, discontinuity)         # Discontinuity point
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1 = line (p1,p2,nelm= n)   # lower boundary
      c2 = translate c4 (p2,-p3)  # right-hand side boundary
      c3 = line (p3,p4,nelm= n)   # upper boundary
      c4 = curves(c11,c12)        # left-hand boundary consisting of two parts
      c11= line (p1,p5,nelm= m1)  # lower part of left-hand boundary
      c12= line (p5,p4,nelm= m2)  # upper part of left-hand boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
                    # Linear triangles are used
```

```
      s1=rectangle3(c1,c2,c3,-c4)

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2

end
```

Since the velocity is a function of the angle, we need a main program

```
      program conv_shock01

!     --- Main program for testing of upwind schemes for 2d convection-diffusion
!         linear triangular elements
!         See manual standard problems, Section 3.1.8.1
!         Shock problem

      call sepcom ( 0 )

      end

!     --- define velocity as function of the angle

      function funccf ( ichoice, x, y, z )
      implicit none
      integer ichoice
      double precision x, y, z, funccf, angle, getconst

!     --- The constant pi is stored in common block consta

      include 'SPcommon/consta'

!     --- angle is defined as a constant

      angle = getconst ( 'angle' )

      if ( ichoice==1 ) then

!     --- ichoice = 1, u = cos(angle)

         funccf = cos(angle/180d0*pi)

      else if ( ichoice==2 ) then

!     --- ichoice = 2, v = sin(angle)

         funccf = sin(angle/180d0*pi)

      else

!     --- Other case, should never be possible

         funccf = 0d0

      end if

      end
```

The corresponding input file is

```
# conv_shock01.prb
#
#  problem file for testing of upwind schemes for 2d convection-diffusion
#  linear triangular elements
#  See manual standard problems, Section 3.1.8.1
#  Shock problem
#
#  To run this file use:
#     sepcomp conv_shock01.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1e-6              # diffusion parameter
      angle     = 45                # angle of velocity
   vector_names
      pot_galerkin
      pot_first_order
      pot_doubly
      pot_dc1
      pot_tri_max
      pot_flip_flop
   variables
      iupwind
      minimum
      maximum
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=800               # Type number for second order elliptic equation
                               # See Standard problems Section 3.1
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      curves(c1)               # Essential boundary conditions on lower boundary
      curves(c4)               # Essential boundary conditions on left-hand side
                               # boundary
end

# Define the essential boundary conditions
#  See Users Manual Section 3.2.5
```

```
essential boundary conditions
   curves(c1) value = 1      # At C3 T=1,
   curves(c11) value = 1     # At C11 T=1,
                             # at C12 we have T=0, which does not require input
end

#  Define the coefficients for Convection-diffusion equation
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients, sequence_number = 1
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
     icoef2  = iupwind           # Type of upwind
      coef6  =  eps               # a11 = eps
      coef9  = coef 6             # a22 = eps
      coef12 = func = 1           # u = cos(angle), see subroutine FUNCCF
      coef13 = func = 2           # v = sin(angle), see subroutine FUNCCF
end

#  Define the structure of the main program
#  See Users Manual Section 3.2.3

structure

   # First case: Galerkin solution
      # Set essential boundary conditions
        prescribe_boundary_conditions pot_galerkin, sequence_number = 1
      # Compute the potential, by solving the linear equations
      # Set the value of the upwind parameter
        iupwind = 0
        solve_linear_system,  pot_galerkin, seq_solve = 1//
           seq_coef = 1
      # Print minimum and maximum of the solution
        minimum = min_max  pot_galerkin, scal_max = maximum
        print 'Galerkin solution'
        print minimum, maximum, text = 'minimum and maximum values'
   # Second case: SUPG first-order solution
      # Set essential boundary conditions
        prescribe_boundary_conditions pot_first_order//
           sequence_number = 1
      # Compute the potential, by solving the linear equations
      # Set the value of the upwind parameter
        iupwind = 1
        solve_linear_system,  pot_first_order, seq_solve = 1//
           seq_coef = 1
      # Print minimum and maximum of the solution
        minimum = min_max  pot_first_order, scal_max = maximum
        print 'SUPG first-order solution'
        print minimum, maximum, text = 'minimum and maximum values'
   # Third case: SUPG doubly assymptotic solution
      # Set essential boundary conditions
        prescribe_boundary_conditions pot_doubly, sequence_number = 1
      # Compute the potential, by solving the linear equations
      # Set the value of the upwind parameter
        iupwind = 3
```

```
        solve_linear_system,  pot_doubly, seq_solve = 1//
           seq_coef = 1
      # Print minimum and maximum of the solution
        minimum = min_max  pot_doubly, scal_max = maximum
        print 'SUPG doubly assymptotic solution'
        print minimum, maximum, text = 'minimum and maximum values'
   # Fourth case: SUPG DC1 solution
      # Set essential boundary conditions
        prescribe_boundary_conditions pot_dc1, sequence_number = 1
      # Compute the potential, by solving the non-linear equations
      # Set the value of the upwind parameter
        iupwind = 7
        solve_nonlinear_system,  pot_dc1, sequence_number = 1
      # Print minimum and maximum of the solution
        minimum = min_max  pot_dc1, scal_max = maximum
        print 'SUPG discontinuity capturing'
        print minimum, maximum, text = 'minimum and maximum values'
   # Fifth case: SUPG triangular elements with maximum principle
   # Underelaxation is applied
      # Set essential boundary conditions
        prescribe_boundary_conditions pot_tri_max, sequence_number = 1
      # Compute the potential, by solving the non-linear equations
      # Set the value of the upwind parameter
      # The iteration is started with the doubly assymptotic solution
        iupwind = 3
        solve_nonlinear_system,  pot_tri_max, sequence_number = 2
      # Print minimum and maximum of the solution
        minimum = min_max  pot_tri_max, scal_max = maximum
        print 'SUPG triangular elements with maximum principle'
        print minimum, maximum, text = 'minimum and maximum values'
   # Sixth case: SUPG triangular elements with maximum principle
   #             suppress flip-flop
      # Set essential boundary conditions
        prescribe_boundary_conditions pot_flip_flop, sequence_number = 1
      # Compute the potential, by solving the non-linear equations
      # Set the value of the upwind parameter
      # The iteration is started with the doubly assymptotic solution
        iupwind = 3
        solve_nonlinear_system,  pot_flip_flop, sequence_number = 3
      # Print minimum and maximum of the solution
        minimum = min_max  pot_flip_flop, scal_max = maximum
        print 'SUPG triangular elements with maximum principle, no flip-flop'
        print minimum, maximum, text = 'minimum and maximum values'

   output

end

# input for non-linear solver
# Input for DC1
nonlinear_equations, sequence_number = 1     # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-3,print_level=2, lin_solver=1//
   at_error return
   equation 1
      fill_coefficients 1
```

```
end

# Input for SUPG triangular elements with maximum principle
nonlinear_equations, sequence_number = 2     # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-5,print_level=2, lin_solver=1//
   at_error return, relaxation = 0.9
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
end

# Input for SUPG triangular elements with maximum principle
# Suppress flip-flop
nonlinear_equations, sequence_number = 3     # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-3,print_level=2, lin_solver=1//
   at_error return
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
         at_iteration 4, sequence_number 3
end


#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number=1 # input for iteration 2
   elgrp1
      icoef2 = 9            # triangular elements with maximum principle
end

change coefficients, sequence_number=2 # input for iteration 3
   elgrp1
      icoef2 = 10           # initialize flip flop array
end

change coefficients, sequence_number=3 # input for iteration 4
   elgrp1
      icoef2 = 11           # update flip flop array
end
```

In order to check the behaviour of the method, we have compared the minimum and maximum values of the solution. This is a measure for the appearance of wiggles.
Table 3.1.8.1 gives these minimum and maximum values for the methods used.

Table 3.1.8.1 Minimum and maximum values of the solution (triangles)

Type of method	minimum value	maximum value
Galerkin	-1.83421E-04	1.33327E+00
SUPG first-order	-3.78362E-02	1.17226E+00
SUPG doubly asymptotic	-3.78362E-02	1.17226E+00
SUPG discontinuity capturing	-8.77571E-04	1.05377E+00
SUPG with maximum principle	0	1
SUPG with maximum principle suppressing flip-flop	0	1

In order to inspect the solution, the following input file for program SEPPOST may be used:

```
#  conv_shock01.pst
#  Input file for postprocessing of upwind schemes for 2d convection-diffusion
#  linear triangular elements
#  See manual standard problems, Section 3.1.8.1
#  Shock problem
#  To run this file use:
#     seppost conv_shock01.pst > conv_shock01.post.out
#
#  Reads the files meshoutput and sepcomp.out
#
postprocessing                              # See Users Manual Section 5.2
   define colour table (1, 6,7,8,9,10,11,12,13,14,15,20)
   plot contour pot_galerkin           # make a contour plot of the potential
   3d plot pot_galerkin, angle = 135   # 3d plot of potential
      plot coloured levels pot_galerkin//
         minlevel = 0, maxlevel = 1, nlevel =12
                                            # coloured level plot of the potential
   plot contour pot_first_order           # make a contour plot of the potential
   3d plot pot_first_order, angle = 135   # 3d plot of potential
      plot coloured levels pot_first_order//
         minlevel = 0, maxlevel = 1, nlevel =12
                                            # coloured level plot of the potential
   plot contour pot_doubly           # make a contour plot of the potential
   3d plot pot_doubly, angle = 135   # 3d plot of potential
      plot coloured levels pot_doubly//
         minlevel = 0, maxlevel = 1, nlevel =12
                                            # coloured level plot of the potential
   plot contour pot_dc1           # make a contour plot of the potential
   3d plot pot_dc1, angle = 135   # 3d plot of potential
      plot coloured levels pot_dc1//
         minlevel = 0, maxlevel = 1, nlevel =12
                                            # coloured level plot of the potential
   plot contour pot_tri_max           # make a contour plot of the potential
   3d plot pot_tri_max, angle = 135   # 3d plot of potential
      plot coloured levels pot_tri_max//
         minlevel = 0, maxlevel = 1, nlevel =12
                                            # coloured level plot of the potential
   plot contour pot_flip_flop           # make a contour plot of the potential
   3d plot pot_flip_flop, angle = 135   # 3d plot of potential
      plot coloured levels pot_flip_flop//
         minlevel = 0, maxlevel = 1, nlevel =12
                                            # coloured level plot of the potential
end
```

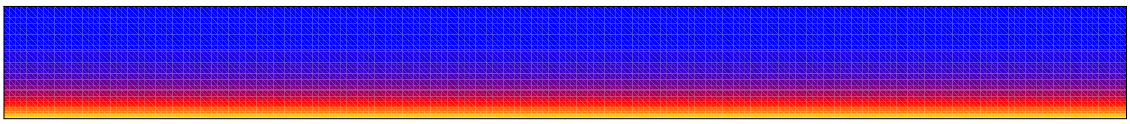


Figure 3.1.7.2: Coloured contour plot of the stable saturation S at $\tau = 5$

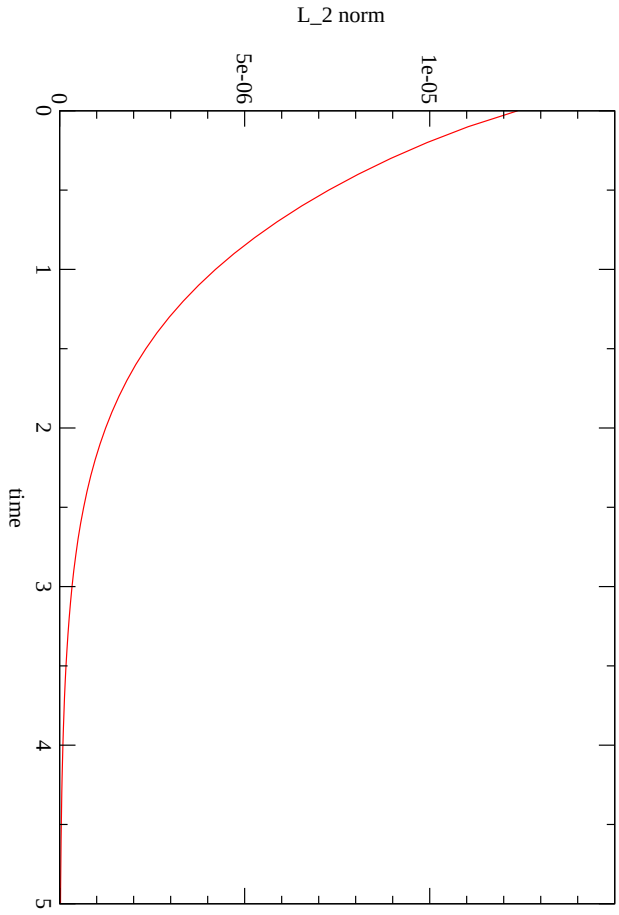


Figure 3.1.7.3: The L_2-norm versus time

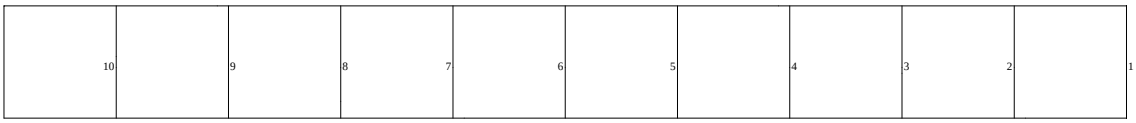


Figure 3.1.7.4: The stream-function Ψ_y at $\tau = 5$

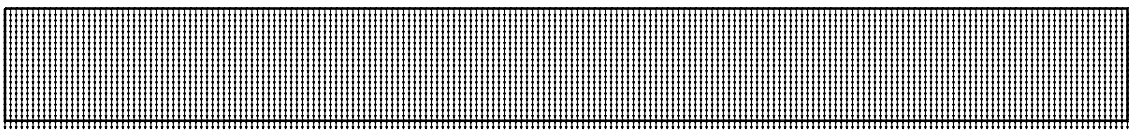


Figure 3.1.7.5: The velocity field $\mathbf{U}$ at $\tau = 5$

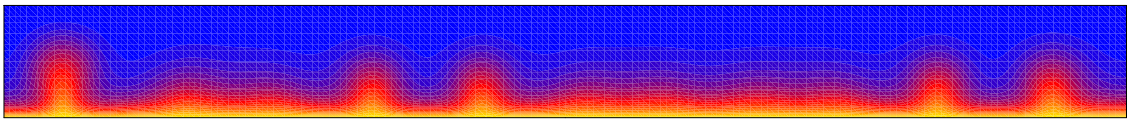


Figure 3.1.7.6: Coloured contour plot of the stable saturation S at $\tau = 5$

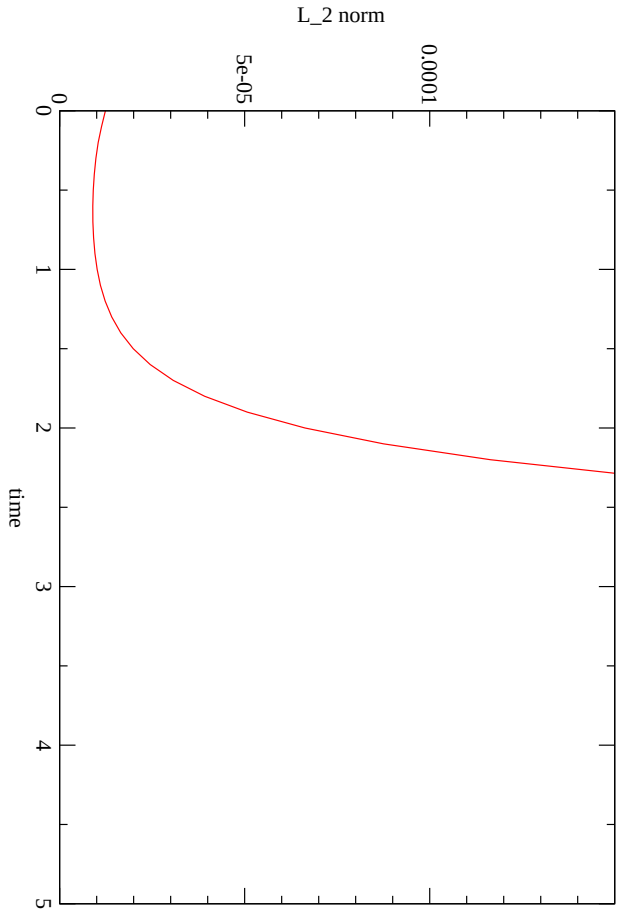


Figure 3.1.7.7: The L_2-norm versus time

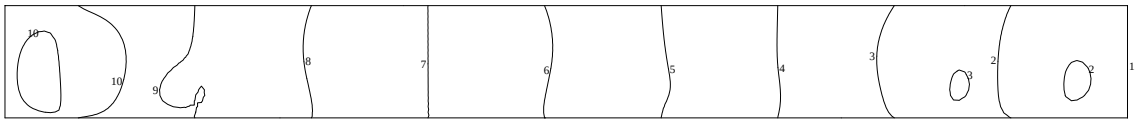


Figure 3.1.7.8: The stream-function Ψ_y at $\tau = 5$

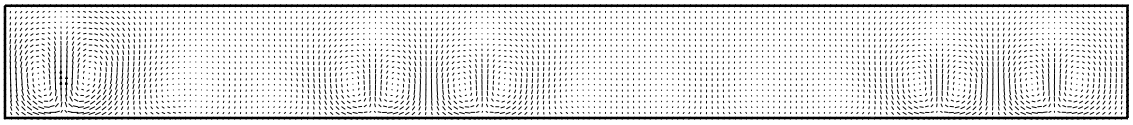


Figure 3.1.7.9: The velocity $\mathbf{U}$ at $\tau = 5$

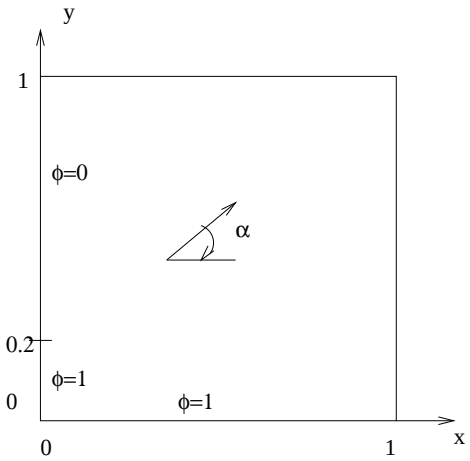


Figure 3.1.8.1: Definition of region for skew convection

Figures 3.1.8.2 to 3.1.8.5 show the three-dimensional representations for the solutions of the Galerkin case, the SUPG case, SUPG with discontinuity capturing and SUPG satisfying the maximum principle respectively.

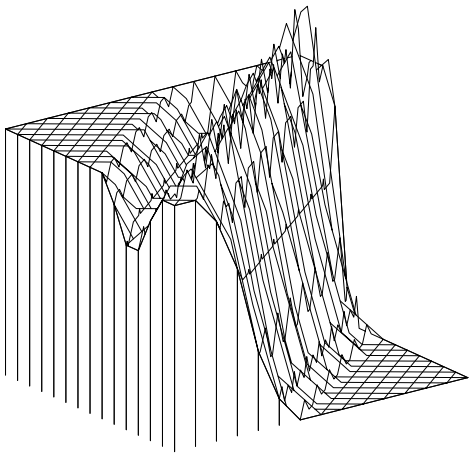


Figure 3.1.8.2: Galerkin solution

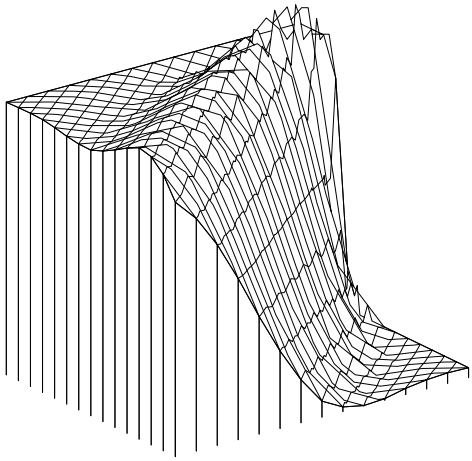


Figure 3.1.8.3: SUPG, first order

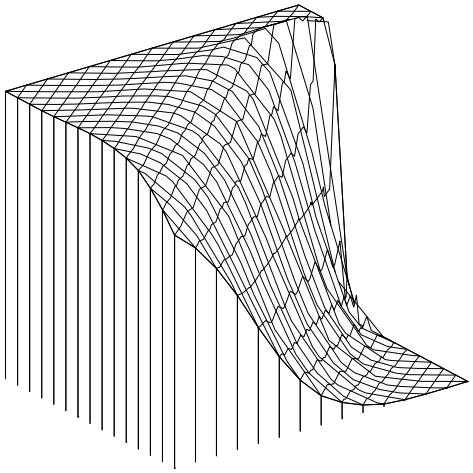


Figure 3.1.8.4: Discontinuity capturing

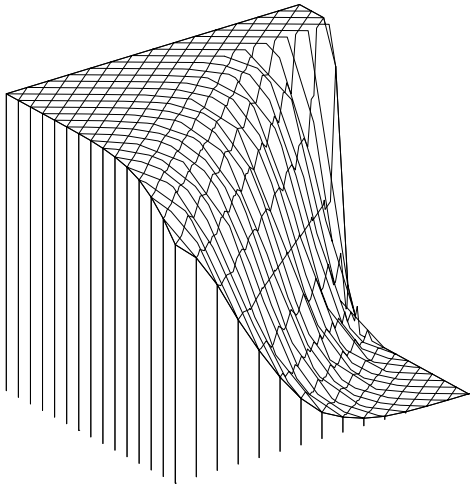


Figure 3.1.8.5: SUPG, satisfying the maximum principle

Figures 3.1.8.6 to 3.1.8.9 show coloured contour levels for the same cases, where black defines the region with values at most equal to 0, and yellow the values larger or equal to 1. All other colours represent values between.
Mark that the yellow colour in the last picture is due to the plot subroutine; all values in the left under corner triangle are exactly equal to 1.

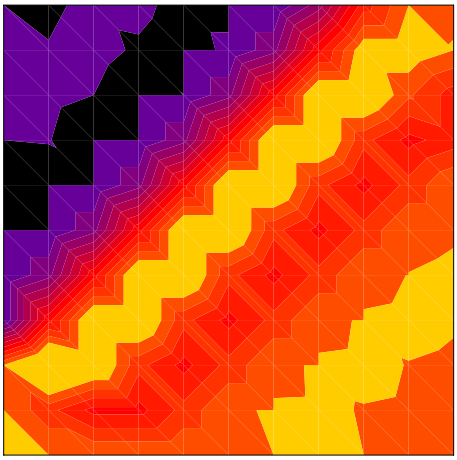


Figure 3.1.8.6: Galerkin solution

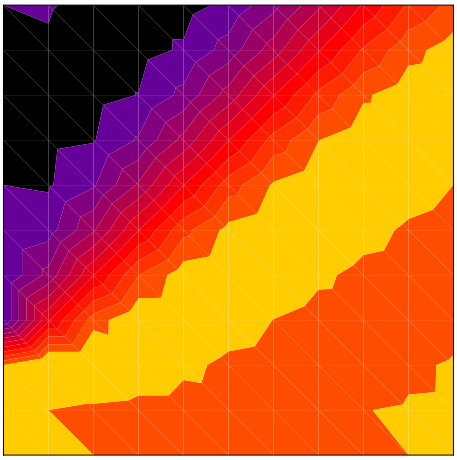


Figure 3.1.8.7: SUPG, first order

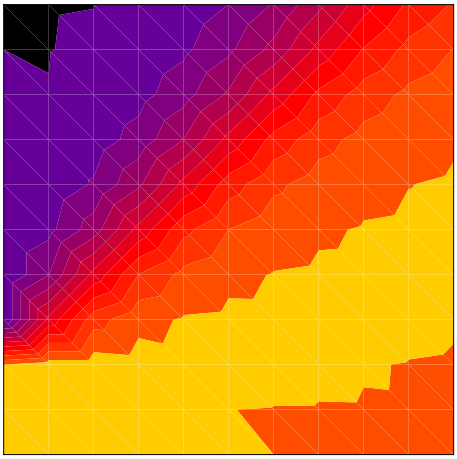


Figure 3.1.8.8: Discontinuity capturing

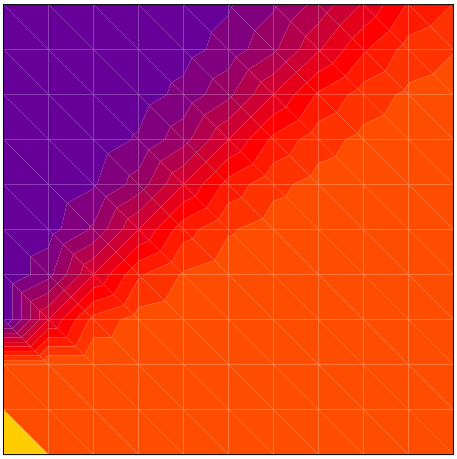


Figure 3.1.8.9: SUPG, satisfying the maximum principle

Table 3.1.8.2 gives these minimum and maximum values for the methods used in case of bilinear quadrilaterals.

Table 3.1.8.2 Minimum and maximum values of the solution (quadrilaterals)

Type of method	minimum value	maximum value
Galerkin	-3.38159E-04	1.37479E+00
SUPG first-order	-3.64150E-02	1.09637E+00
SUPG doubly asymptotic	-3.64150E-02	1.09637E+00
SUPG discontinuity capturing	0	1.00003E+00

The result of the discontinuity capturing is reached after 6 iterations. Increasing the accuracy would lead to a smaller maximum value and more iterations.

3.1.8.2 Rotating cone problem

In this example we consider the so-called rotating cone problem. Consider the square Ω: (-0.5,-0.5) × (0.5,0.5) drawn in Figure 3.1.8.10. From the centre to the mid point of the under boundary a

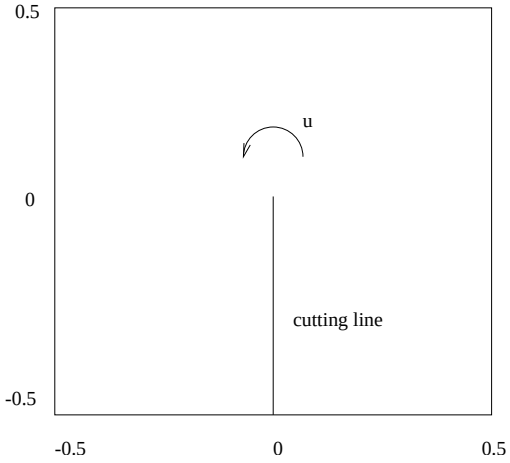


Figure 3.1.8.10: Definition of region for rotating cone problem

cut C is defined. We assume that we have to solve the convection-diffusion equation:

$$-\varepsilon\Delta c + \mathbf{u}\cdot c = 0$$

The parameter ε is chosen equal to 10^{-6}, which means that we are nearly dealing with pure convection equation. At the outer boundary we impose the Dirichlet boundary condition $c = 0$. The velocity vector $\mathbf{u}$ is equal to $(-y, x)$, which implies that the flow rotates around the centroid counterclockwise. At the inflow side of the cut C the concentration c is given by a Gauss curve: $c = cos(2\pi(y + 0.25))$. At the outflow part of the cut C no boundary condition is given, which means that implicitly the boundary condition $\frac{\partial c}{\partial n} = 0$ is imposed.
Due to the small amount of diffusion the Gauss curve should be rotated without any damping and the value of c at the outflow part of the cut must be nearly identical to that at the inflow part.

To get this example into your local directory use:

```
sepgetex rotatxx
```

with `xx` equal to 01 02 03 or 04.
These options correspond to the following cases:

01 Linear triangular elements

02 Linear triangular elements with limiting

03 Bi-linear quadrilaterals

04 Bi-linear quadrilaterals with limiting

To run these problems use:

```
sepmesh rotatxx.msh
sepview sepplot.001
seplink rotatxx
rotatxx < rotatxx.prb
seppost rotatxx.pst
sepview sepplot.001
```

The mesh input file for the linear triangle case is given by

```
#  rotat01.msh
#
#  mesh file for testing of upwind schemes for 2d convection-diffusion
#  linear triangular elements
#  See manual standard problems, Section 3.1.8.2
#  Rotating cone problem
#
#  To run this file use:
#     sepmesh rotat01.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      xmin = -0.5             # minimum x-value
      xmax =  0.5             # maximum x-value
      ymin = -0.5             # minimum y-value
      ymax =  0.5             # maximum y-value
   integers
      n  = 10          # number of elements along one half of a side
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=( xmin, ymin)              # Left under point
      p2=( xmax, ymin)              # Right under point
      p3=( xmax, ymax)              # Right upper point
      p4=( xmin, ymax)              # Left upper point
      p5=(0,0)                      # centroid
      p10=(0, ymin)                 # centre of lower side (left part)
      p11=(0, ymin)                 # centre of lower side (right part)
      p12=(0, ymax)                 # centre of upper side
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1 = line (p1,p10,nelm= n)  # lower boundary (left part)
      c2 = line (p10,p5,nelm= n)  # cutting line (left part)
      c3 = line (p5,p12,nelm= n)  # artificial line from centroid to
                                  # upper boundary
      c11 = curves(c2,c3)         # artificial line from lower boundary to
                                  # upper boundary (left part)
      c4 = translate c1 (p4,p12)  # upper boundary (left part)
      c5 = translate c11(p1,-p4)  # left-hand boundary
                                  # the minus sign is used to indicate the end
                                  # point
```

```
      c6 = line (p11,p2,nelm= n)  # lower boundary (right part)
      c7 = translate c11(p2,-p3)  # right-hand boundary
                                  # the minus sign is used to indicate the end
                                  # point
      c8 = translate c6 (p12,p3)  # upper boundary (right part)
      c9 = line (p11,p5,nelm= n)  # cutting line (right part)
      c12 = curves(c9,c3)         # artificial line from lower boundary to
                                  # upper boundary (right part)
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
                    # Linear triangles are used
      s1=rectangle3(c1,c11,-c4,-c5)  # left-hand part
      s2=rectangle3(c6,c7,-c8,-c12)  # right-hand part

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2

end
```

Since the velocity and the boundary conditions are a function of the coordinates, we need a main program.

```
      program rotat01

!     --- Main program for testing of upwind schemes for 2d convection-diffusion
!         linear triangular elements
!         See manual standard problems, Section 3.1.8.2
!         Rotating cone problem

      call sepcom ( 0 )

      end

!     --- define velocity as function of the co-ordinates

      function funccf ( ichoice, x, y, z )
      implicit none
      integer ichoice
      double precision x, y, z, funccf

      if ( ichoice==1 ) then

!     --- ichoice = 1, u = -y

         funccf = -y

      else if ( ichoice==2 ) then

!     --- ichoice = 2, v = x

         funccf = x

      else
```

```
!     --- Other case, should never be possible

         funccf = 0d0

      end if

      end

!     --- define concentration as boundary condition on curve c2

      function funcbc ( ichoice, x, y, z )
      implicit none
      integer ichoice
      double precision x, y, z, funcbc

!     --- The constant pi is stored in common block consta

      include 'SPcommon/consta'


      if ( ichoice==1 ) then

!     --- ichoice = 1, c = cos(2pi (y+0.25))

         funcbc = cos(2d0*pi*(y+0.25d0))

      else

!     --- Other case, should never be possible

         funcbc = 0d0

      end if

      end
```

The corresponding input file is

```
# rotat01.prb
#
#  problem file for testing of upwind schemes for 2d convection-diffusion
#  linear triangular elements
#  See manual standard problems, Section 3.1.8.2
#  Rotating cone problem
#
#  To run this file use:
#     sepcomp rotat01.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
```

```
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1e-6              # diffusion parameter
   vector_names
      pot_galerkin
      pot_first_order
      pot_doubly
      pot_dc1
      pot_tri_max
      pot_flip_flop
   variables
      iupwind
      minimum
      maximum
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=800                # Type number for second order elliptic equation
                                # See Standard problems Section 3.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      curves(c1)                # Essential boundary conditions on left part
                                # of lower boundary
      curves(c4 to c8)          # Essential boundary conditions on all other
                                # outer boundaries
      curves(c9)                # Essential boundary conditions on right part
                                # of cutting line
end

# Define the essential boundary conditions
#  See Users Manual Section 3.2.5

essential boundary conditions
   curves(c9) func = 1       # At C9 the concentration is a function
end

#  Define the coefficients for Convection-diffusion equation
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients, sequence_number = 1
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
     icoef2  = iupwind           # Type of upwind
      coef6  =  eps               # a11 = eps
      coef9  = coef 6             # a22 = eps
      coef12 = func = 1           # u = cos(angle), see subroutine FUNCCF
      coef13 = func = 2           # v = sin(angle), see subroutine FUNCCF
end
```

```
#  Define the structure of the main program
#  See Users Manual Section 3.2.3

structure

   # First case: Galerkin solution
      # Set essential boundary conditions
        prescribe_boundary_conditions pot_galerkin, sequence_number = 1
      # Compute the potential, by solving the linear equations
      # Set the value of the upwind parameter
        iupwind = 0
        solve_linear_system,  pot_galerkin, seq_solve = 1//
           seq_coef = 1
      # Print minimum and maximum of the solution
        minimum = min_max  pot_galerkin, scal_max = maximum
        print 'Galerkin solution'
        print minimum, maximum, text = 'minimum and maximum values'
   # Second case: SUPG first-order solution
      # Set essential boundary conditions
        prescribe_boundary_conditions pot_first_order//
           sequence_number = 1
      # Compute the potential, by solving the linear equations
      # Set the value of the upwind parameter
        iupwind = 1
        solve_linear_system,  pot_first_order, seq_solve = 1//
           seq_coef = 1
      # Print minimum and maximum of the solution
        minimum = min_max  pot_first_order, scal_max = maximum
        print 'SUPG first-order solution'
        print minimum, maximum, text = 'minimum and maximum values'
   # Third case: SUPG doubly assymptotic solution
      # Set essential boundary conditions
        prescribe_boundary_conditions pot_doubly, sequence_number = 1
      # Compute the potential, by solving the linear equations
      # Set the value of the upwind parameter
        iupwind = 3
        solve_linear_system,  pot_doubly, seq_solve = 1//
           seq_coef = 1
      # Print minimum and maximum of the solution
        minimum = min_max  pot_doubly, scal_max = maximum
        print 'SUPG doubly assymptotic solution'
        print minimum, maximum, text = 'minimum and maximum values'
   # Fourth case: SUPG DC1 solution
      # Set essential boundary conditions
        prescribe_boundary_conditions pot_dc1, sequence_number = 1
      # Compute the potential, by solving the non-linear equations
      # Set the value of the upwind parameter
        iupwind = 7
        solve_nonlinear_system,  pot_dc1, sequence_number = 1
      # Print minimum and maximum of the solution
        minimum = min_max  pot_dc1, scal_max = maximum
        print 'SUPG discontinuity capturing'
        print minimum, maximum, text = 'minimum and maximum values'
   # Fifth case: SUPG triangular elements with maximum principle
```

```
   # Underelaxation is applied
      # Set essential boundary conditions
        prescribe_boundary_conditions pot_tri_max, sequence_number = 1
      # Compute the potential, by solving the non-linear equations
      # Set the value of the upwind parameter
      # The iteration is started with the doubly assymptotic solution
        iupwind = 3
        solve_nonlinear_system,  pot_tri_max, sequence_number = 2
      # Print minimum and maximum of the solution
        minimum = min_max  pot_tri_max, scal_max = maximum
        print 'SUPG triangular elements with maximum principle'
        print minimum, maximum, text = 'minimum and maximum values'
   # Sixth case: SUPG triangular elements with maximum principle
   #             suppress flip-flop
      # Set essential boundary conditions
        prescribe_boundary_conditions pot_flip_flop, sequence_number = 1
      # Compute the potential, by solving the non-linear equations
      # Set the value of the upwind parameter
      # The iteration is started with the doubly assymptotic solution
        iupwind = 3
        solve_nonlinear_system,  pot_flip_flop, sequence_number = 3
      # Print minimum and maximum of the solution
        minimum = min_max  pot_flip_flop, scal_max = maximum
        print 'SUPG triangular elements with maximum principle, no flip-flop'
        print minimum, maximum, text = 'minimum and maximum values'

   output

end

# input for non-linear solver
# Input for DC1
nonlinear_equations, sequence_number = 1     # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-3,print_level=2, lin_solver=1//
   at_error return
   equation 1
      fill_coefficients 1
end

# Input for SUPG triangular elements with maximum principle
nonlinear_equations, sequence_number = 2     # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-5,print_level=2, lin_solver=1//
   at_error return, relaxation = 0.9
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
end

# Input for SUPG triangular elements with maximum principle
# Suppress flip-flop
nonlinear_equations, sequence_number = 3     # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-3,print_level=2, lin_solver=1//
   at_error return
   equation 1
```

```
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
         at_iteration 4, sequence_number 3
end


#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number=1 # input for iteration 2
   elgrp1
      icoef2 = 9            # triangular elements with maximum principle
end

change coefficients, sequence_number=2 # input for iteration 3
   elgrp1
      icoef2 = 10           # initialize flip flop array
end

change coefficients, sequence_number=3 # input for iteration 4
   elgrp1
      icoef2 = 11           # update flip flop array
end
```

In order to check the behaviour of the method, we have compared the minimum and maximum values of the solution. This is a measure for the appearance of wiggles.
Table 3.1.8.3 gives these minimum and maximum values for the methods used.

Table 3.1.8.3 Minimum and maximum values of the solution (triangles)

Type of method	minimum value	maximum value
Galerkin	-6.64338E-02	1.06728E+00
SUPG first-order	-1.35177E-02	1.00493E+00
SUPG doubly asymptotic	-1.35177E-02	1.00493E+00
SUPG discontinuity capturing	0	1
SUPG with maximum principle	0	1
SUPG with maximum principle suppressing flip-flop	0	1

Both the method with discontinuity capturing and with the maximum principle get a divergence message after 5 iterations. Carefully playing with underrelaxation may improve this behaviour but it is hard to get real convergence. The flip-flop method behaves the best in this case, although the final solution does not have a better quality.
In order to inspect the solution, the following input file for program SEPPOST may be used:

```
#  rotat01.pst
#  Input file for postprocessing of upwind schemes for 2d convection-diffusion
#  linear triangular elements
#  See manual standard problems, Section 3.1.8.2
#  Rotating cone problem
#  To run this file use:
#     seppost rotat01.pst > rotat01.post.out
#
#  Reads the files meshoutput and sepcomp.out
#
```

```
postprocessing                              # See Users Manual Section 5.2
   define colour table (1, 6,7,8,9,10,11,12,13,14,15,20)
   plot contour pot_galerkin           # make a contour plot of the potential
   3d plot pot_galerkin, angle = 135   # 3d plot of potential
      plot coloured levels pot_galerkin//
         minlevel = 0, maxlevel = 1, nlevel =12
                                          # coloured level plot of the potential
   plot contour pot_first_order           # make a contour plot of the potential
   3d plot pot_first_order, angle = 135   # 3d plot of potential
      plot coloured levels pot_first_order//
         minlevel = 0, maxlevel = 1, nlevel =12
                                          # coloured level plot of the potential
   plot contour pot_doubly           # make a contour plot of the potential
   3d plot pot_doubly, angle = 135   # 3d plot of potential
      plot coloured levels pot_doubly//
         minlevel = 0, maxlevel = 1, nlevel =12
                                          # coloured level plot of the potential
   plot contour pot_dc1           # make a contour plot of the potential
   3d plot pot_dc1, angle = 135   # 3d plot of potential
      plot coloured levels pot_dc1//
         minlevel = 0, maxlevel = 1, nlevel =12
                                          # coloured level plot of the potential
   plot contour pot_tri_max           # make a contour plot of the potential
   3d plot pot_tri_max, angle = 135   # 3d plot of potential
      plot coloured levels pot_tri_max//
         minlevel = 0, maxlevel = 1, nlevel =12
                                          # coloured level plot of the potential
   plot contour pot_flip_flop          # make a contour plot of the potential
   3d plot pot_flip_flop, angle = 135   # 3d plot of potential
      plot coloured levels pot_flip_flop//
         minlevel = 0, maxlevel = 1, nlevel =12
                                          # coloured level plot of the potential
end
```

Figures 3.1.8.11 to 3.1.8.14 show the three-dimensional representations for the solutions of the Galerkin case, the SUPG case, SUPG with discontinuity capturing and SUPG satisfying the maximum principle respectively. From these pictures it is clear that the non-linear methods do not

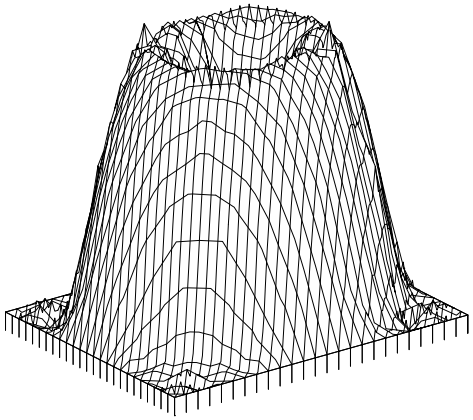


Figure 3.1.8.11: Galerkin solution

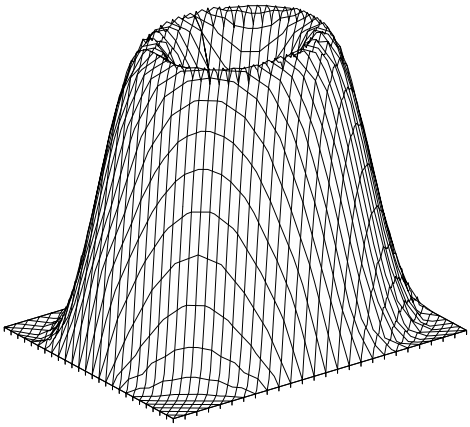


Figure 3.1.8.12: SUPG, first order

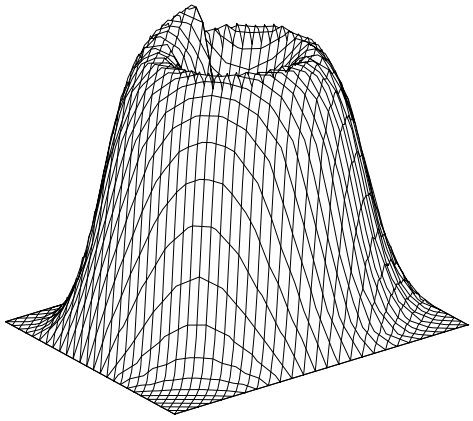


Figure 3.1.8.13: Discontinuity capturing

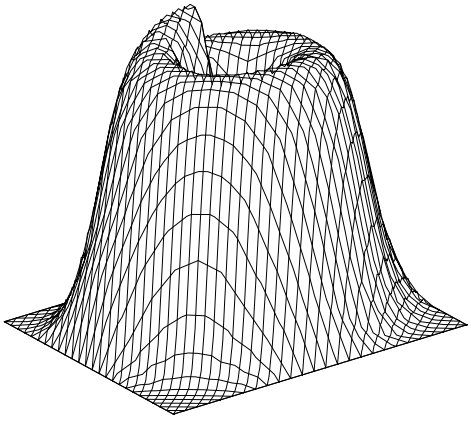


Figure 3.1.8.14: SUPG, satisfying the maximum principle and flip-flop

have values below 0 and above 1, but that the value of the concentration at the cutting line at outflow is considerably smaller than 1. So these methods suffer from crosswind diffusion.

Figures 3.1.8.15 to 3.1.8.18 show coloured contour levels for the same cases, where black defines the region with values at most equal to 0, and yellow the values larger or equal to 1. All other colours represent values between.

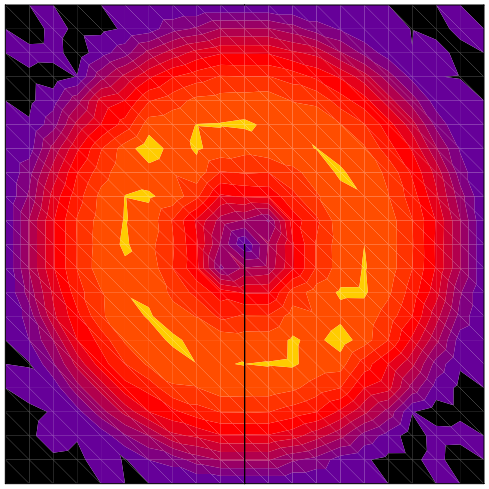


Figure 3.1.8.15: Galerkin solution

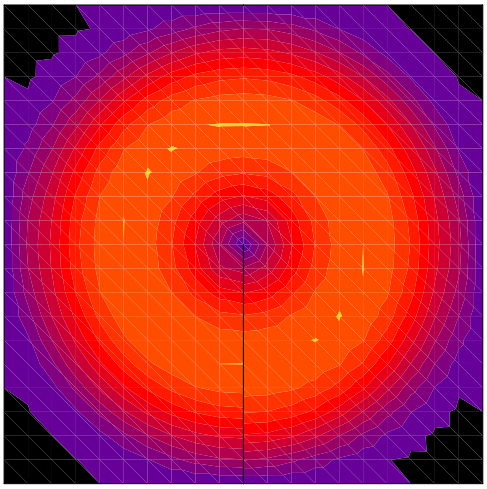


Figure 3.1.8.16: SUPG, first order

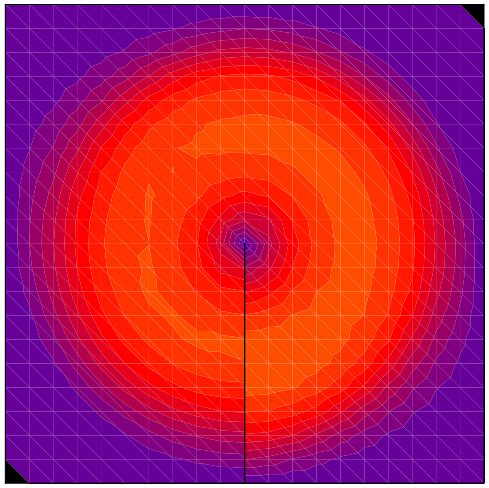


Figure 3.1.8.17: Discontinuity capturing

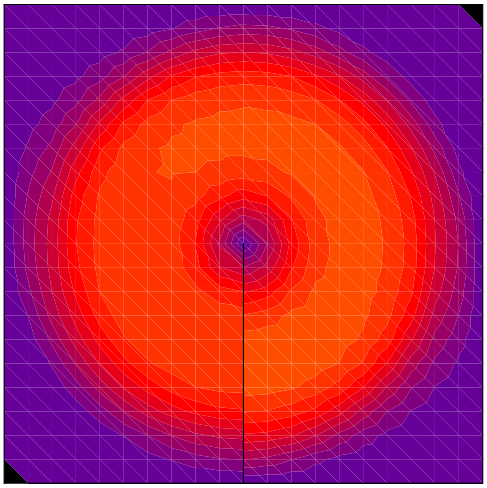


Figure 3.1.8.18: SUPG, satisfying the maximum principle and flip-flop

Table 3.1.8.4 gives these minimum and maximum values for the methods used in case of bilinear quadrilaterals.

Table 3.1.8.4 Minimum and maximum values of the solution (quadrilaterals)

Type of method	minimum value	maximum value
Galerkin	-2.81864E-02	1.03667E+00
SUPG first-order	-7.26277E-03	1.00024E+00
SUPG doubly asymptotic	-7.26277E-03	1.00024E+00
SUPG discontinuity capturing	0	1

The result of the discontinuity capturing is reached after 6 iterations, in which case divergence is discovered. However, the quality of the result is comparable with the triangular mesh.

3.1.9 Some examples of the use of periodical boundary conditions

In this section we give a number of artificial examples, to show the various possibilities of the use of periodical boundary conditions. It concerns the following possibilities

3.1.9.1 Standard periodical boundary conditions

3.1.9.2 Periodical boundary conditions with jump

3.1.9.3 Periodical boundary conditions with multiplication factor

3.1.9.1 Standard periodical boundary conditions

In order to get this example into your local directory use:

```
sepgetex testperiod06
```

To run this example use

```
sepmesh testperiod06.msh
view mesh by jsepview
seplink testperiod06
testperiod06 < testperiod06.prb
view results by jsepview
```

In this example we consider the following artificial problem.
Let Ω be the unit square ((0,1) $\times$ (0,1)).
Let T satisfy the standard Laplace equation, i.e $-\Delta T = 0$.
On the lower boundary ($y = 0$) and the upper boundary ($y = 1$), we prescribe the temperature T by $T(x, y) = sin(2\pi x)$ (Dirichlet boundary condition).
Furthermore on the left-hand and the right-hand side we assume periodical boundary conditions, hence $T_{\text{left}} = T_{\text{right}}$, and $\frac{\partial T}{\partial x}|_{\text{left}} = \frac{\partial T}{\partial x}|_{\text{right}}$.

The equation itself is standard, and so are the Dirichlet boundary conditions. The periodical boundary conditions, however, require so-called connection elements, which identify unknowns on left-hand side and right-hand side. This coupling of unknowns is actually carried out if elements of type -1 are used.
The mesh file used in this case is:

```
#  testperiod06.msh
#
#  mesh file for 2d periodical boundary conditions problem
#  See testperiod06.prb and the manual Examples Section 3.1.9
#  for a description
#
#  To run this file use:
#     sepmesh testperiod06.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1        # width of the region
      length = 1       # length of the region
```

```
   integers
      n = 40           # number of elements in length direction
      m = 10           # number of elements in width direction
      shape_cur = 1    # Linear elements along curves
      shape_sur = 5    # Bi-linear quadrilaterals in surfaces
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)              # Left under point
      p2=( length,0)        # Right under point
      p3=( length, width)   # Right upper point
      p4=(0, width)         # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1=line  shape_cur (p1,p2,nelm= n)     # lower wall
      c2=line  shape_cur (p2,p3,nelm= m)     # right-hand side
      c3=line  shape_cur (p3,p4,nelm= n)     # upper wall
      c4=line  shape_cur (p4,p1,nelm= m)     # left-hand side
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle  shape_sur (c1,c2,c3,c4)

   plot                          # make a plot of the mesh
                                 # See Users Manual Section 2.2

end
```

Since the boundary conditions depend on the coordinates, we need a main program to define the function.

```
      program testperiod06
      implicit none

!     --- File for  2d periodical boundary conditions problem
!         See testperiod06.prb and the manual Examples Section 3.1.9
!         for a description

      call startsepcomp
      end

!     --- Function funcbc for the essential boundary conditions

      function funcbc ( ichoice, x, y, z )
      implicit none
      integer ichoice
```

```
      double precision x, y, z, funcbc
      include 'SPcommon/consta'  ! Contains the value of pi

      if ( ichoice==1 ) then

!     --- ichoice = 1, standard case

         funcbc = sin(2d0*pi*x)

      else

!     --- ichoice # 1, error

         call eropen('funcbc')
         call errint(ichoice,1)
         call errsub ( 1, 1, 0, 0)
         call erclos('funcbc')
         call instop
         funcbc = 0d0

      end if

      end
```

The input file for the computational part is standard. The only special part is the definition of the periodical boundary conditions.

```
# testperiod06.prb
#
#  problem file for 2d periodical boundary conditions problem
#  See manual Examples Section 3.1.9
#
#  The problem to be solved consist of a square of size 1x1:
#  S: (0,0) x (1,1)
#
#  The equation to be solved is the standard Laplacian equation
#  The boundary conditions at lower and upper wall are given by sin(2 pi x)
#  On the left-hand and right-hand sides we have periodical boundary conditions,
#  hence
#  T_left = T_right
#  dT/dx_left =d T/dx_right
#
#  To run this file use:
#     sepcomp testperiod06.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      kappa     = 1                 # conductivity
```

```
   vector_names
      Temperature
end
#
#  Define the type of problem to be solved
#
problem                    # See Users Manual Section 3.2.2

   types                         # Define types of elements,
                                 # See Users Manual Section 3.2.2
      elgrp1=800                 # Type number for second order elliptic equation
                                 # See Standard problems Section 3.1
   essbouncond                   # Define where essential boundary conditions are
                                 # given (not the value)
                                 # See Users Manual Section 3.2.2
      curves(c1)                 # Fixed under wall
      curves(c3)                 # Fixed upper wall
   periodical_boundary-conditions
      curves(c2,-c4)
end
#
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#
structure                     # See Users Manual Section 3.2.3
   matrix_structure symmetric
 # Compute the temperature
   prescribe_boundary_conditions, vector = Temperature func=1, curves(c1)
   prescribe_boundary_conditions, vector = Temperature func=1, curves(c3)
   solve_linear_system, vector = Temperature
   print Temperature
   plot_colored_levels Temperature
   output
end
#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients
   elgrp1
      coef6  =  kappa      # 6:  Heat conduction
      coef9  = coef6       # 9:  Heat conduction
end

end_of_sepran_input
```

Figure 3.1.9.1.1 shows the computed temperature.

3.1.9.2 Periodical boundary conditions with jump

The second example is almost identical to the first one, with the exception of the boundary conditions. The Dirichlet boundary conditions in this case are $T = x$ and in the periodical boundary conditions we have a jump of size 1, hence $T_{\text{right}} = T_{\text{left}} + 1$, and $\frac{\partial T}{\partial x}|_{\text{right}} = \frac{\partial T}{\partial x}|_{\text{left}}$.

In order to get this example into your local directory use:

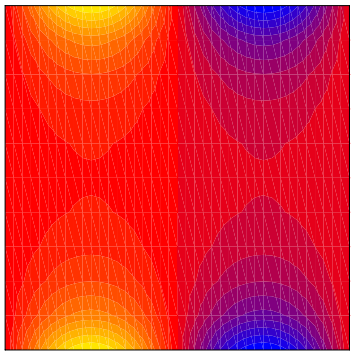


Figure 3.1.9.1: Coloured contour plot of Temperature

```
sepgetex testperiod07
```

To run this example use

```
sepmesh testperiod07.msh
view mesh by jsepview
seplink testperiod07
testperiod07 < testperiod07.prb
view results by jsepview
```

The mesh file in this case is identical to that in Subsection 3.1.9.1.1, except that in the connection elements c2 and c4 are interchanged. The fortran file requires an extra function `func` to define the exact solution. The problem file is a little bit different because of the jump and since the exact solution is compared with the computed one. The error is printed. The error appears to be of the order of the machine precision. Also the postprocessing file is the same as for the first example. For completeness we give the problem file.

```
# testperiod07.prb
#
#  problem file for 2d periodical boundary conditions problem
#  See manual Examples Section 3.1.9
#
#  The problem to be solved consist of a square of size 1x1:
#  S: (0,0) x (1,1)
#
#  The equation to be solved is the standard Laplacian equation
#  The boundary conditions at lower and upper wall are given by x
#  On the left-hand and right-hand sides we have periodical boundary conditions.
#  Special in this case is that there is constant jump of size 1 between
#  right-hand side and left-hand side, hence
#  T_right = T_left + 1
#  dT/dx_left =d T/dx_right
#
#  One can verify that the exact solution is given by T = x
#  To run this file use:
#     sepcomp testperiod07.prb
#
#  Reads the file meshoutput
```

```
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      kappa     = 1                 # conductivity
   vector_names
      Temperature
      T_exact
   variables
      error
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=800                # Type number for second order elliptic equation
                                # See Standard problems Section 3.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      curves(c1)                # Fixed under wall
      curves(c3)                # Fixed upper wall
   periodical_boundary-conditions
      curves(c4,-c2) constant = 1
end
#
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#
structure                    # See Users Manual Section 3.2.3
 #  Define the structure of the large matrix
   matrix_structure symmetric
 # Compute the Temperature
   prescribe_boundary_conditions, Temperature func=1, curves(c1)
   prescribe_boundary_conditions, Temperature func=1, curves(c3)
   solve_linear_system, Temperature
 # Create the exact solution
   create_vector T_exact func=1
 # Compute and print the error
   error = norm_dif=3, vector1= Temperature, vector2= T_exact
   print error
 # Write the results to a file
   output
   plot_colored_levels Temperature
end

#  Define the coefficients for the problems
#  All parameters not mentioned are zero
```

```
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients
   elgrp1
      coef6  =  kappa      # 6:  Heat conduction
      coef9  = coef6       # 9:  Heat conduction
end

end_of_sepran_input
```

3.1.9.3 Periodical boundary conditions with multiplication factor

exception of the boundary conditions and the right-hand side. The Dirichlet boundary conditions in this case are $T = 3 + 4x - x^2$ and in the periodical boundary conditions we have a multiplication factor of size 2, hence
$T_{\text{right}} = 2T_{\text{left}}$, and $2\frac{\partial T}{\partial x}|_{\text{right}} = \frac{\partial T}{\partial x}|_{\text{left}}$.
Furthermore the source term in the Poisson equation is equal to 2, hence we solve $-\Delta T = 2$.

In order to get this example into your local directory use:

```
sepgetex testperiod09
```

To run this example use

```
sepmesh testperiod09.msh
view mesh by jsepview
seplink testperiod09
testperiod09 < testperiod09.prb
view results by jsepview
```

The mesh file in this case is identical to that in Subsection 3.1.9.2.2. The fortran file requires an extra function `func` to define the exact solution. The problem file is a little bit different because of the multiplication factor and the source term.

Mark that in this case the multiplication factor in the boundary condition in combination with the requirement $2\frac{\partial T}{\partial x}|_{\text{right}} = \frac{\partial T}{\partial x}|_{\text{left}}$ make the boundary condition periodical.

The error appears to be of the order of the machine precision. Also the postprocessing file is the same as for the first example. For completeness we give the problem file.

```
# testperiod09.prb
#
#  problem file for 2d periodical boundary conditions problem
#  See manual Examples Section 3.1.9
#
#  The problem to be solved consist of a square of size 1x1:
#  S: (0,0) x (1,1)
#
#  The equation to be solved is the standard Poisson equation with rhs 2
#  The boundary conditions at lower and upper wall are given by x
#  On the left-hand and right-hand sides we have periodical boundary conditions.
#  Special in this case is that there is multiplication factor of size 2 between
#  right-hand side and left-hand side, hence
#  T_right = 2 T_left
#  Furthermore the derivatives are different
#  dT/dx_left = 2 d T/dx_right
```

```
#
#  One can verify that the exact solution is given by T = 3+4x-x^2
#  To run this file use:
#     sepcomp testperiod09.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      kappa     = 1                 # conductivity
   vector_names
      Temperature
      T_exact
   variables
      error
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=800               # Type number for second order elliptic equation
                               # See Standard problems Section 3.1
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      curves(c1)               # Fixed under wall
      curves(c3)               # Fixed upper wall
   periodical_boundary-conditions
      curves(c4,-c2) factor = 2
end
#
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#
structure                   # See Users Manual Section 3.2.3
 # Define structure of matrix
   matrix_structure, symmetric
 # Compute the Temperature
   prescribe_boundary_conditions, Temperature, func=1, curves(c1,c3)
   solve_linear_system, Temperature
 # Create the exact solution
   create_vector T_exact func=1
 # Compute and print the error
   error = norm_dif=3, vector1=Temperature, vector2=T_exact
   print error
 # Write the results to a file
   output
```

```
end

#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients
   elgrp1
      coef6  =  kappa      # 6:  Heat conduction
      coef9  = coef6       # 9:  Heat conduction
      coef16 = 2           # 16: Source term
end

end_of_sepran_input
```

3.1.10 Some examples of the use of periodical boundary conditions to connect two regions

In this section we give a number of artificial examples, to show how periodical boundary conditions can be used to connect two regions through boundary conditions.
It concerns the following possibilities

3.1.10.1 Standard periodical boundary conditions

3.1.10.2 Periodical boundary conditions with multiplication factor

3.1.10.1 Standard periodical boundary conditions

In order to get this example into your local directory use:

```
sepgetex testperiod03
```

To run this example use

```
sepmesh testperiod03.msh
view mesh by jsepview
seplink testperiod03
testperiod03 < testperiod03.prb
view results by jsepview
```

In this example we consider the following artificial problem.
Let Ω_1 be the unit square ((0,1) $\times$ (0,1)) and Ω_2 be the unit square ((1,1) $\times$ (2,1))
Let T satisfy the diffusion equation with different diffusion parameters κ in each region, i.e $-$ div $\kappa_1 \nabla T = 0$ in Ω_1 and $-$ div $\kappa_2 \nabla T = 0$ in Ω_2.
On the lower boundary ($y = 0$) and the upper boundary ($y = 1$), as well as the left-hand side of Ω_1 and the right-hand side of Ω_2 we prescribe the temperature T by $T(x, y) = sin(2\pi x)$ (Dirichlet boundary condition).
Furthermore we assume that both regions which have separate boundaries for $x = 1$ are coupled through coupling conditions. The number of coupling conditions must be the same as for periodical boundary conditions. Since we are dealing with a second order equation with one unknown it is necessary to prescribe exactly one condition on each boundary. This means that on the connecting boundary we need two boundary conditions (one for each curve).
The boundary conditions we prescribe are continuity of T and that the flux that goes from Ω_1 is equal to the flux that enters Ω_2 through the curves at $x = 1$.
So if the curves at $x = 1$ are defined as Cleft and Cright, actually the boundary condition is defined as $T_{\text{Cleft}} = T_{\text{Cright}}$ and $\kappa_1 \frac{\partial T}{\partial x}|_{\text{Cleft}} = \kappa_2 \frac{\partial T}{\partial x}|_{\text{Cright}}$. These are exactly the periodical boundary conditions

The equation itself is standard, and so are the Dirichlet boundary conditions. The periodical boundary conditions, however, require so-called connection elements, which identify unknowns on Cleft and Cright. This coupling of unknowns is actually carried out if elements of type -1 are used.
The mesh file used in this case is:

```
#  testperiod03.msh
#
#  mesh file for 2d periodical boundary conditions problem
#  See testperiod03.prb for a description
#
#  To run this file use:
#     sepmesh testperiod03.msh
```

```
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1        # width of the region
      length = 1       # length of the first subregion
      length2 = 2      # length of the second subregion
   integers
      n = 20           # number of elements in length direction
      m = 10           # number of elements in width direction
      shape_cur = 1    # Linear elements along curves
      shape_sur = 5    # Bi-linear quadrilaterals in surfaces
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
     # subregion 1
      p1=(0,0)              # Left under point
      p2=( length,0)        # Right under point
      p3=( length, width)   # Right upper point
      p4=(0, width)         # Left upper point
     # subregion 2
      p11=( length,0)         # Left under point
      p12=( length2,0)        # Right under point
      p13=( length2, width)   # Right upper point
      p14=( length, width)    # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
     # subregion 1
      c1=line  shape_cur (p1,p2,nelm= n)     # lower wall
      c2=line  shape_cur (p2,p3,nelm= m)     # right-hand side
      c3=line  shape_cur (p3,p4,nelm= n)     # upper wall
      c4=line  shape_cur (p4,p1,nelm= m)     # left-hand side
     # subregion 2
      c11=line  shape_cur (p11,p12,nelm= n)     # lower wall
      c12=line  shape_cur (p12,p13,nelm= m)     # right-hand side
      c13=line  shape_cur (p13,p14,nelm= n)     # upper wall
      c14=line  shape_cur (p14,p11,nelm= m)     # left-hand side
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
     # subregion 1
      s1=rectangle  shape_sur (c1,c2,c3,c4)
     # subregion 2
```

```
      s2=rectangle  shape_sur (c11,c12,c13,c14)

#  Coupling of surfaces to element groups

   meshsurf
      selm1 = s1
      selm2 = s2

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2
end
```

Since the boundary conditions depend on the coordinates, we need a main program to define the function.

```
      program testperiod03
      implicit none

!     --- File for  2d periodical boundary conditions problem
!         See testperiod03.prb and the manual Examples Section 3.1.10
!         for a description

      call startsepcomp
      end

!     --- Function funcbc for the essential boundary conditions

      function funcbc ( ichoice, x, y, z )
      implicit none
      integer ichoice
      double precision x, y, z, funcbc
      include 'SPcommon/consta'  ! Contains the value of pi

      if ( ichoice==1 ) then

!     --- ichoice = 1, Omega_1

         funcbc = sin(2d0*pi*x)

      else if ( ichoice==3 ) then

!     --- ichoice = 3, Omega_2

         funcbc = sin(2d0*pi*x)

      else

!     --- ichoice # 1,3: error

         call eropen('funcbc')
         call errint(ichoice,1)
         call errsub ( 1, 1, 0, 0)
         call erclos('funcbc')
         call instop
         funcbc = 0d0
```

```
      end if

      end

!     --- Function func for the creation of the exact solution

      function func ( ichoice, x, y, z )
      implicit none
      integer ichoice
      double precision x, y, z, func, funcbc

      func = funcbc ( ichoice, x, y, z )

      end
```

The input file for the computational part is standard. The only special part is the formed by the definition of the periodical boundary conditions.

```
# testperiod03.prb
#
#  problem file for 2d periodical boundary conditions problem
#  See manual Examples Section 3.1.10
#
#  The problem to be solved consist of two squares of size 1x1:
#  S1: (0,0) x (1,1)
#  S2: (1,0) x (2,1)
#
#  The squares are connected by connection elements
#
#  In S1 the solution of the diffusion equation is: T = sin(2 pi x)
#  In S2 the solution of the diffusion equation is: T = sin(2 pi x)
#
#  The coefficients for the diffusion equation are different for both squares
#
#  To run this file use:
#     sepcomp testperiod03.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      kappa_1 = 1                 # conductivity in S1
      kappa_2 = 2                 # conductivity in S2
   vector_names
      Temperature
end
#
#  Define the type of problem to be solved
#
```

```
problem                    # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=800                # Type number for second order elliptic equation
                                # See Standard problems Section 3.1
      elgrp2=800                # Type number for second order elliptic equation
                                # See Standard problems Section 3.
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      curves(c1)                # Fixed under wall S1
      curves(c3)                # Fixed upper wall S1
      curves(c4)                # left-hand side S1
      curves(c11)               # Fixed under wall S2
      curves(c13)               # Fixed upper wall S2
      curves(c12)               # left-hand side S2
                                # All not prescribed boundary conditions
                                # satisfy corresponding stress is zero
   periodical_boundary_conditions
      curves(c2,-c14)
end
#
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#
structure                    # See Users Manual Section 3.2.3
 # Compute the temperature
   prescribe_boundary_conditions, Temperature &
      degfd1, func=1, curves(c1 to c4)
   prescribe_boundary_conditions, Temperature &
      degfd1, func=3, curves(c11 to c14)  # curve c14 has no effect
   solve_linear_system, Temperature
   print Temperature
   plot_colored_levels Temperature
 # Write the results to a file
   output
end

#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients
   elgrp1
      coef6  =  kappa_1    # 6:  Heat conduction
      coef9  = coef6       # 9:  Heat conduction
   elgrp2
      coef6  =  kappa_2    # 6:  Heat conduction
      coef9  = coef6       # 9:  Heat conduction
end

end_of_sepran_input
```

Figure 3.1.10.1.1 shows the computed temperature.

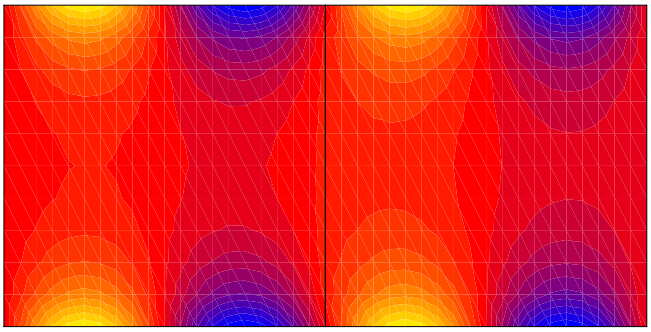


Figure 3.1.10.1: Coloured contour plot of Temperature

3.1.10.2 Periodical boundary conditions with multiplication factor

The second example is almost identical to the first one, with the exception of the boundary conditions. The Dirichlet boundary conditions in this case are $T = y$ in Ω_1 and $T = 2y$ in Ω_2. In the periodical boundary conditions we have a multiplication factor of size 2, hence $T_{\text{right}} = 2T_{\text{left}}$, and $2\kappa_2 \frac{\partial T}{\partial x}|_{\text{right}} = \kappa_1 \frac{\partial T}{\partial x}|_{\text{left}}$.

In order to get this example into your local directory use:

```
sepgetex testperiod02
```

To run this example use

```
sepmesh testperiod02.msh
view mesh by jsepview
seplink testperiod02
testperiod02 < testperiod02.prb
view results by jsepview
```

The mesh file in this case is identical to that in Subsection 3.1.10.1.1 The fortran file requires an extra function `func` to define the exact solution. The problem file is a little bit different because of the multiplication factor, the source term and the symmetry.

Mark that in this case the matrix is symmetrical due to the multiplication factor in the periodical boundary condition in combination with the requirement $2\kappa_2 \frac{\partial T}{\partial x}|_{\text{right}} = \kappa_1 \frac{\partial T}{\partial x}|_{\text{left}}$.

The error appears to be of the order of the machine precision. Also the postprocessing file is the same as for the first example. For completeness we give the problem file.

```
# testperiod02.prb
#
#  problem file for 2d periodical boundary conditions problem
#  problem is stationary and linear
#
#  The problem to be solved consist of two squares of size 1x1:
#  S1: (0,0) x (1,1)
#  S2: (1,0) x (2,1)
#
```

```
#  The squares are connected by connection elements
#
#  In S1 the solution of the laplacian equation is: T = y
#  In S2 the solution of the laplacian equation is: T = 2y
#  Hence in the common interface we have a multiplication factor of 2 for T
#
#  The coefficients for the diffusion equation are different for both squares
#  In this case we use the symmetric solution method
#
#  To run this file use:
#     sepcomp testperiod02.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      kappa_1 = 1                 # conductivity in S1
      kappa_2 = 2                 # conductivity in S2
   vector_names
      Temperature
      T_exact
   variables
      error
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=800               # Type number for double laplacian equation
                               # See Standard problems Section 3.5
      elgrp2=800               # Type number for double laplacian equation
                               # See Standard problems Section 3.5
                               # the multiplication factor 2
                               # may be used for connection elements only
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      curves(c1)               # Fixed under wall
      curves(c3)               # Fixed side walls and instream boundary
      curves(c4)               # inflow
      curves(c11)              # Fixed under wall
      curves(c13)              # Fixed side walls and instream boundary
      curves(c12)              # Outstream boundary (v-component given)
                               # All not prescribed boundary conditions
                               # satisfy corresponding stress is zero
   periodical_boundary-conditions
      curves(c2,-c14) degfd1, constant = 0, factor = 2   # The jump is 0,
```

```
                              # the multiplication factor 2
end
#
#
structure                    # See Users Manual Section 3.2.3
 # Define structure of matrix
   matrix_structure, symmetric
 # Compute the Temperature (vector 1)
   prescribe_boundary_conditions, Temperature, func=1, curves(c1 to c4)
     # curve c2 has no effect
   prescribe_boundary_conditions, Temperature, func=3, curves(c11 to c14)
     # curve c14 has no effect
   create_vector vector= T_exact degfd1, func=1, surface(s1)
   create_vector vector= T_exact degfd1, func=3, surface(s2)
   solve_linear_system,  Temperature
   error = norm_dif=3, vector1= Temperature, vector2= T_exact
   plot_colored_levels Temperature
 # Write the results to a file
   output
   print error
end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients
   elgrp1
      coef6  =  kappa_1    # 6:  Heat conduction
      coef9  = coef6       # 9:  Heat conduction
   elgrp2
      coef6  =  kappa_2    # 6:  Heat conduction
      coef9  = coef6       # 9:  Heat conduction
end

end_of_sepran_input
```

3.1.11 Experiments with the shifted Laplace operator to solve the real Helmholtz equation

The Helmholtz equation is usually the result of putting solution of the form e^{ikt} into the wave equation, where k is the wave number. A simple example a such a Helmholtz equation is given by

$$-\Delta\phi - k^2\phi = f \tag{3.1.11.1}$$

If k is large, the corresponding discretization matrix is indefinite. As a result, iterative linear solvers do not converge, or converge very slowly.
A possibility to improve the convergence of such a solver is the use of a so-called shifted Laplace preconditioner. This preconditioner is not based on the original equation (3.1.11.1), but on the following shifted equation

$$-\Delta\phi + \beta k^2\phi = f, \tag{3.1.11.2}$$

with β some non-negative parameter.
The corresponding matrix is positive definite and hence the construction of an ILU preconditioner based on this matrix does not introduce any difficulties. It appears that this shifted Laplace ILU preconditioner may improve the convergence of iterative methods considerably for a well chosen value of β. Mark that $\beta = -1$ corresponds to a standard ILU preconditioner.
In this section we solve Equation (3.1.11.1) on the domain $\Omega = [0,1]^2$ with boundary conditions $\phi = 0$ everywhere. The function f is chosen equal to $-(k^2 - 5\pi^2)sin(\pi x)sin(2\pi y)$.

To get this example into your local directory use:

```
sepgetex helmholtz1x
```

with x equal to 1 or 2, where 1 refers to the classical method and 2 to the shifted Laplace preconditioner. and to run it use:

```
sepmesh helmholtz1x.msh
seplink helmholtz1x
helmholtz1x < helmholtz1x.prb
```

The input file for the mesh is very simple:

```
#  helmholtz11.msh
#
#  mesh file for the example as described in Section 3.1.11 of
#  the manual Examples
#
#  To run this file use:
#     sepmesh helmholtz11.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants            # See Users Manual Section 1.4
   integers
      n = 50
   reals
      width = 1         # width of the region
      heigth = 1        # heigth of the region
end
#
```

```
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)
      p2=(width,0)
      p3=(width,heigth)
      p4=(0,heigth)
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1 = line (p1,p2,nelm=n)
      c2 = line (p2,p3,nelm=n)
      c3 = line (p3,p4,nelm=n)
      c4 = line (p4,p1,nelm=n)
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1 = rect3 (c1,c2,c3,c4)
   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2

end
```

It is clear that linear triangles are used. To compute the real error made by the iterative solver, we first solve the equations by a direct solver (profile method) and afterwards by the iterative solver and subtract both solutions to get the error. In case of `helmholtz11` we use BICGSTAB as solver with ILU preconditioner. The required accuracy is 10^{-4} and by setting the print level to 2, we are able to follow the convergence of the iteration process.
The corresponding input file is:

```
# helmholtz11.prb
#
#  problem file for the example as described in Section 3.1.11 of
#  the manual Examples
#  The Helmholtz equation is solved by a BiCgstab method with ILU preconditioner
#
#  To run this file use:
#     sepcomp helmholtz11.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
set warn off  # suppress warnings
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
```

```
      mu = 1                  # permeability
      k = 10                  # wave number
      beta = -k^2             # coefficient for the zeroth order term
   vector_names
      potential               # solution of the iterative solver
      potex                   # solution computed by the direct solver
      diff                    # difference between potential and poted
   variables
      error                   # error made by the iterative solver
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1 = (type=800)      # Type number for Poisson equation
                               # See Standard problems Section 3.1
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      curves (c1 to c4)        # whole boundary
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix, sequence_number = 1
   storage_scheme = profile   # storage scheme for the direct solver
end
matrix, sequence_number = 2
   storage_scheme = compact   # storage scheme for the iterative solver
end
#
#  The coefficients for the differential equation
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1
#
coefficients
   elgrp1
      coef 6 =  mu         # Constant permeability
      coef 9 =  coef 6     # Constant permeability
      coef15 =  beta       # coefficient for the zeroth order term, defined
                           # by the wave number
      coef16 =  func=1     # the right-hand side is a function of space
end
#
#  Linear solver
#  See Users Manual, Section 3.2.8
#
solve, sequence_number = 1  ! use direct method
  # no input required
end
solve, sequence_number = 2  ! use iterative method (bicgstab with ILU precon)
```

```
   iteration_method = cg, preconditioner = ilu, print_level = 2, eps = 1d-4
end

structure

  # First we compute potex by a direct solver

   prescribe_boundary_conditions potex  ! no input required, since the value is 0
   solve_linear_system potex            ! computes potex

  # next we compute potential by the iterative solver
  # It is necessary to change the structure of the matrix

   change_structure_of_matrix, seq_structure = 2
   prescribe_boundary_conditions potential       ! no input required
   solve_linear_system potential, seq_solve = 2  ! computes potential
   diff = potential - potex                      ! difference between both
   error = norm=3, diff                          ! norm of difference
   print error
end

end_of_sepran_input
```

Since the right-hand side is a function of x and y we need a function subroutine `funccf` and hence a main program `helmholtz11`, given by:

```
      program helmholtz11

!     --- Standard main program

      implicit none
      integer, allocatable, dimension (:) :: ibuffr
      integer pbuffr, error
      parameter ( pbuffr=100000000)
      allocate(ibuffr(pbuffr), stat = error)
      if (error /= 0) then
         ! space for these arrays could not be allocated
         print *, "error: (helmholtz11) could not allocate space."
         stop
      end if  ! (error /= 0)
      call sepcombf ( ibuffr, ibuffr, pbuffr )
      end

!     --- Function funccf is used to define the right-hand side

      function funccf ( ichoice, x, y, z )
      implicit none
      integer ichoice
      double precision x, y, z, funccf
      include 'SPcommon/consta'
      double precision k, getconst
      k = getconst('k')
      if ( ichoice==1 ) then
         funccf = -(k**2-5*pi**2)*sin(pi*x)*sin(2d0*pi*y)
      else
```

```
         call errchr('funccf',1)
         call errsub ( 349, 0, 0, 1)
         call instop
      end if
      end
```

For the shifted Laplace operator we can use the same mesh file and program. The problem files changes only in the matrix input block and the solve input block. Below we give the changed input blocks:

```
matrix, sequence_number = 2
   storage_scheme = compact, shifted_laplace # storage scheme for the iterative solver
end

solve, sequence_number = 2  ! use iterative method (bicgstab with ILU precon)
   iteration_method = cg, preconditioner = ilu, print_level = 2, eps = 1d-4 //
      laplace_shift = 1
end
```

Table (3.1.11.1) shows the number of iterations required to solve Equation (3.1.11.2) by a standard Bi-Cgstab method, and for the shifted preconditioner for a shift equal to zero and one equal to one. The number of nodes is equal to n^2, where n takes the values 50, 100 and 150. The wave number, k, varies from 10 to 40. A dash in the column means that the iteration process does not converge. From the table it is clear that the gain for the shifted Laplace preconditioner is large for the combination large wave number and smaller number of elements. Increasing the number of nodes, or decreasing the wave number increases the condition of the matrix and makes it more suitable for standard iterative solvers.

Table 3.1.11.1 Number of iterations for several values of the shift

		Bi-CGstab			shift 0			shift 1		
n		50	100	150	50	100	150	50	100	150
k	10	38	60	84	44	54	84	44	56	80
	20	72	56	62	60	58	58	58	62	50
	30	236	48	30	58	36	32	32	28	36
	40	-	64	30	44	32	24	16	22	20

3.2 Second order complex linear elliptic and parabolic equations with one degree of freedom

In this section we treat the following examples of real elliptic and parabolic equations with one degree of freedom.

3.2.1 An artificial mathematical example, just to show how to solve a complex elliptic equation.

3.2.2 Experiments with the shifted Laplace operator to solve the complex Helmholtz equation.

3.2.1 An artificial mathematical example

A simple model for wind generated movements in a harbor using a complex potential approach, is given by the model equation:

$$\Delta\phi + K^2\phi + \gamma ik\phi = 0 \tag{3.2.1.1}$$

with

$k^2 = \frac{\omega^2}{gh}$,

$\omega = \frac{2\pi}{T}$,

g the acceleration of gravity,

h the depth of the harbor,

γ the friction coefficient and

T the wave period.

It has been assumed that the depth of the harbor is uniform and that the wave period T is rather large; T $>$ 15 seconds.

To get this example into your local directory use:

```
sepgetex exam3-2-1
```

and to run it use:

```
sepmesh exam3-2-1.msh
seplink exam3-2-1
exam3-2-1 < exam3-2-1.prb
seppost exam3-2-1.pst
```

We consider a very simple model of a rectangular harbor: $0 \leq x \leq L$, $0 \leq y \leq B$.
Figure 3.2.1.1 shows a sketch of the harbor with corresponding definition of points and curves.

In this problem we shall use the following data:

L = 2000 m

B = 690 m

g = 9.81 m/s^2

T = 112 s

ω = 0.056 s^{-1}

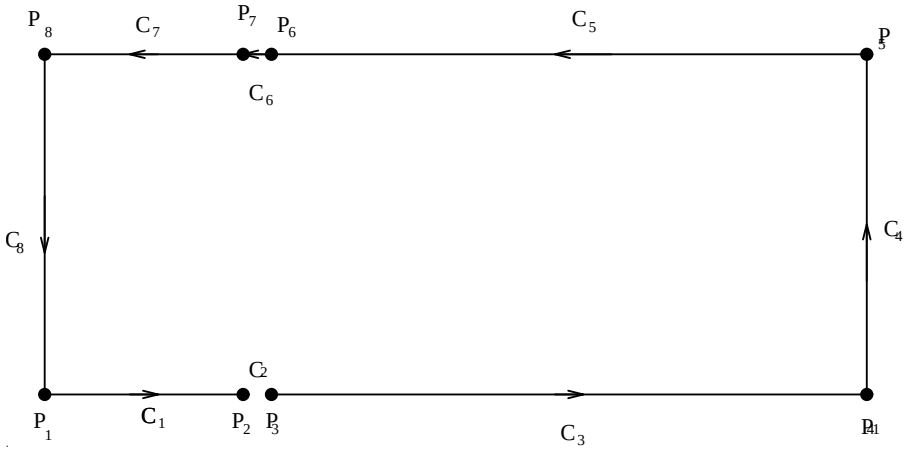


Figure 3.2.1.1: Harbor with fixed boundaries C1-C5, C7, C8 and open boundary C6

h = 15 m

For a unique solution of the problem it is necessary to give boundary conditions for all boundaries. Suppose that the incoming waves have an angle of incidence of 270° to the entrance of the harbor and assume that all closed boundaries of the harbor give total reflection, i.e. $\frac{\partial \phi}{\partial n} = 0$.
Assume that an essential boundary condition of the form $\phi = e^{ik(xcos(\alpha)+ysin(\alpha)}$ is given at the open boundary, with α the angle of incidence, and (x, y) the Cartesian co-ordinates.

The region is subdivided into triangles by the submesh generator "RECTANGLE". As an example linear triangles have been used.
SEPMESH needs an input file. An example of an input file for this region is given below:

```
*****************************************************************************
*
*      File:  exam3-2-1.msh
*
*      Contents:  Mesh for the example 3-2-1 in the manual examples
*
*****************************************************************************
*
mesh2d
   points
      p1 = ( -427.5 ,   0 )
      p2 = (  -41.5 ,   0 )
      p3 = (   42.5 ,   0 )
      p4 = ( 1642.5 ,   0 )
      p5 = ( 1642.5 , 690 )
      p6 = (   41.5 , 690 )
      p7 = (  -42.5 , 690 )
      p8 = ( -427.5 , 690 )
   curves
      c1 = line1 ( p1, p2, nelm=2 )
      c2 = line1 ( p2, p3, nelm=2 )
      c3 = line1 ( p3, p4, nelm=6 )
      c4 = line1 ( p4, p5, nelm=6 )
      c5 = line1 ( p5, p6, nelm=6 )
      c6 = line1 ( p6, p7, nelm=2 )
      c7 = line1 ( p7, p8, nelm=2 )
```

```
      c8 = line1 ( p8, p1, nelm=6 )
      c9 = curves ( c1, c2, c3 )
      c10= curves ( c5, c6, c7 )
   surfaces
      s1 = rectangle3 ( c9, c4, c10, c8 )
   plot ( plotfm=10 )
end
```

Figure 3.2.1.2 shows the mesh generated by SEPMESH.

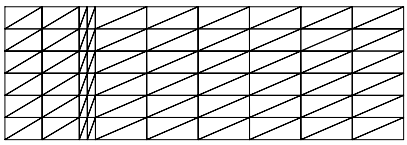


Figure 3.2.1.2: Plot of mesh generated by SEPMESH

The internal elements are defined by type number 150. Only the coefficients 1, 3 and 6 have to be defined; 1 and 3 get the value 1, β is a function defined by the subroutine cfuncf.
The boundary conditions at side C6 are essential boundary conditions, the boundary conditions at the other sides are natural boundary conditions requiring no boundary elements at all. Since in this case it is necessary to define a function subroutine for the coefficient β and the essential boundary condition, it is not possible to use the standard program SEPCOMP. We shall give here the simple program based upon sepcomp and extended with the subroutines CFUNCF and CFUNCB.

First we give the program that is based upon SEPCOMP. The main program consists only of a call to SEPCOM. The listing for this program is given by:

```
! ************************************************************************
!
!      File:  exam3-2-1.f
!
!      Contents:  Main program for the test example described
!                 in the SEPRAN manual examples 3-2-1
!                 Waves in harbour
!                 Since a function subroutine is used for the solution,
!                 it is not possible to use sepcomp
!
```

```
!      Usage:     Compile and link this program with the SEPRAN libraries
!                 seplink exam3-2-1
!                 Run this program with input: exam3-2-1.prb
!
!                 exam3-2-1 < exam3-2-1.prb > exam3-2-1.out
!
!      version 1.0    date   17-06-94
!
! ************************************************************************
!
      program exam321
      implicit none
      double precision pi, omega, g, angle
      common / hacons / pi, omega, g, angle
      pi = 4*atan(1.0d0)
      g = 9.81d0
      angle = 270d0
      omega = 0.056d0
      call sepcom (0)
      end


!     --- subroutine cfuncb for the definition of the boundary conditions


      subroutine cfuncb ( ichois, x, y, z, comval )
      implicit none
      complex * 16 comval
      integer ichois
      double precision x, y, z, depth, alpha, ak, arg

      double precision pi, omega, g, angle
      common / hacons / pi, omega, g, angle

!    --- compute boundary condition in nodal point (x, y, z) :
!        determine wave-number ak :

      depth = 15.0d0

!     angle of incoming wave :

      alpha = (pi/180.0d0)*angle
      ak = omega/sqrt(g*depth)
      arg = ak*(x*cos(alpha)+y*sin(alpha))

!     prescribed elevation :

      comval = dcmplx ( cos(arg), sin(arg) )

      end


!     --- subroutine cfuncf for the definition of the coefficients

      subroutine cfuncf ( ichois, x, y, z, comval )
```

```
      implicit none
      complex * 16 comval
      integer ichois
      double precision x, y, z, gamma, depth, ak

      double precision pi, omega, g, angle
      common / hacons / pi, omega, g, angle

      gamma = 0d0
      depth = 15.0d0
      ak = omega/sqrt(g*depth)

      if ( ichois==1 ) then
          comval = dcmplx ( -ak*ak, -gamma*ak/depth )
      end if
      end
```

This program needs an input file which is the same as for SEPCOMP. The following input file may be used to solve the problem:

```
*****************************************************************************
*
*      File:  exam3-2-1.prb
*
*      Contents:  Input for program exam3-2-1 described in section 3-2-1 in
*                 the manual examples
*                 Waves in harbour
*                 The standard sepcomp approach is used
*
*****************************************************************************
*

constants
   vector_names
      complex_potential
      amplitude_potential
      phase_potential
end

*  Problem definition
*

problem
   types
      elgrp1=(type=150)
   essbouncond
      curves (c6)
end

*  Since special vectors are required at output, it is necessary to
*  define the structure of the program

structure
```

```
   prescribe_boundary_conditions, complex_potential
   solve_linear_system, seq_coef=1, seq_solve=1, complex_potential
   amplitude_potential = modulus complex_potential
   phase_potential = phase complex_potential
   output

end

* Define essential boundary conditions

essential complex boundary conditions
   func = 1                   # Use subroutine cfuncb
end

* Definition of coefficients

complex coefficients
   elgrp 1 ( nparm = 7 )      # Internal element has 7 coefficients
      coef 1 = 1              # a11 = 1
      coef 3 = coef 1         # a22 = a11
      coef 6 = (func=1)       # beta is function given by cfuncf
                              # All other coefficients are 0
end

*  Definition of matrix structure
#complex symmetrical matrix, direct solution method

matrix
   symmetric, complex
end

end_of_sepran_input
```

Once the solution has been computed, it may be printed and plotted by the postprocessing program SEPPOST. SEPPOST also requires an input file. The following input file prints the computed solution, makes a standard contour plot as well as a colored contour plot. In order to identify the plot an extra text identification is submitted.

```
************************************************************************
*
*      File:  exam3-2-1.pst
*
*      Contents:  Input for the postprocessing part of the example described
*                 in Section 3.2.1 of the manual examples
*                 Waves in harbour
*
*      Usage:     seppost exam3-2-1.pst > exam3-21.out
************************************************************************
*
postprocessing
   print amplitude_potential
   print phase_potential
   plot identification, text='waves in harbour', origin=(3,18)
   define plot parameters = norotate
   plot contour amplitude_potential, minlevel=-1, maxlevel=1, nlevel=11
```

```
   plot contour phase_potential, nlevel=12
   3d plot complex_potential, degfd=1, angle=135, lindirec=3//
   text = '3D plot of elevation'
end
```

Figure 3.2.1.3 shows the contour plot of the magnitude made by program SEPPOST. This plot may be visualized by the program SEPDISPLAY.

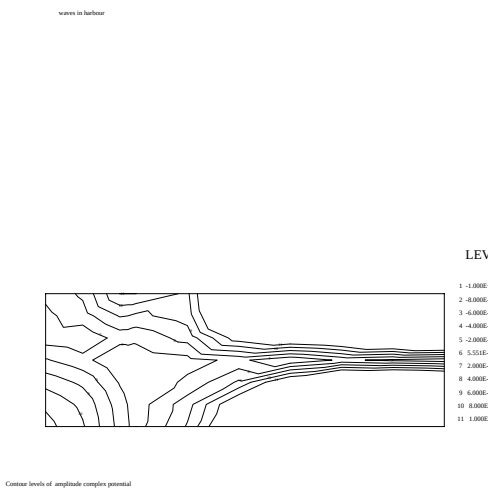


Figure 3.2.1.3: Contour plot of magnitude generated by SEPPOST

Figures 3.2.1.4 and 3.2.1.5 show the contour plot of the phase and the three-dimensional plot of the elevation respectively.

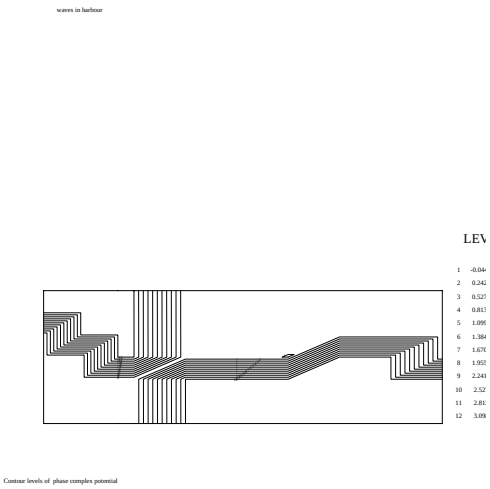


Figure 3.2.1.4: Contour plot of phase

3.2.2 Experiments with the shifted Laplace operator to solve the complex Helmholtz equation

The example treated here is exactly the same as the one in Section (3.1.11). The only difference is that the complex Helmholtz equations are used, which means that the coefficients may be complex and that the Laplace shift may be complex. In this case we use real coefficients for the complex case but consider a complex shift.

To get this example into your local directory use:

```
sepgetex helmholtz2x
```

with x equal to 1 or 2, where 1 refers to the classical method and 2 to the shifted Laplace preconditioner. and to run it use:

```
sepmesh helmholtz2x.msh
seplink helmholtz2x
helmholtz2x < helmholtz2x.prb
```

The input file for the mesh is exactly the same as in Section (3.1.11). The problem file differs only slightly from the real case. Below we give the complete input.

```
# helmholtz21.prb
#
#  problem file for the example as described in Section 3.2.2 of
#  the manual Examples
#  The Helmholtz equation is solved by a BiCgstab method with ILU preconditioner
#
```

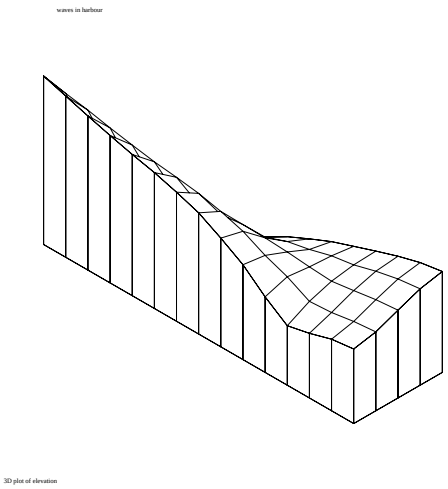


Figure 3.2.1.5: 3D plot of elevation

```
#  To run this file use:
#     sepcomp helmholtz21.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
set warn off  # suppress warnings
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      mu = 1                 # permeability
      k = 10                 # wave number
      beta = -k^2            # coefficient for the zeroth order term
   vector_names
      potential              # solution of the iterative solver
      potex                  # solution computed by the direct solver
      diff                   # difference between potential and poted
   variables
      error                  # error made by the iterative solver
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2
```

```
   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1 = (type=150)       # Type number for complex Helmholtz equation
                                # See Standard problems Section 3.2
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      curves (c1 to c4)         # whole boundary
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix, sequence_number = 1
   storage_scheme = profile, complex   # storage scheme for the direct solver
end
matrix, sequence_number = 2
   storage_scheme = compact, complex   # storage scheme for the iterative solver
end
#
#  The coefficients for the differential equation
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1
#
complex coefficients
   elgrp1
      coef 1 =  mu          # Constant permeability
      coef 3 =  coef 1      # Constant permeability
      coef 6 =  beta        # wave number
      coef 7 =  func=1      # the right-hand side is a function of space
end
#
#  Linear solver
#  See Users Manual, Section 3.2.8
#
solve, sequence_number = 1  ! use direct method
  # no input required
end
solve, sequence_number = 2  ! use iterative method (bicgstab with ILU precon)
   iteration_method = cg, preconditioner = ilu, print_level = 2, eps = 1d-4
end

structure

  # First we compute potex by a direct solver

   prescribe_boundary_conditions potex  ! no input required, since the value is 0
   solve_linear_system potex            ! computes potex

  # next we compute potential by the iterative solver
  # It is necessary to change the structure of the matrix

   change_structure_of_matrix, seq_structure = 2
   prescribe_boundary_conditions potential       ! no input required
   solve_linear_system potential, seq_solve = 2  ! computes potential
```

```
   diff = potential - potex                    ! difference between both
   error = norm=3, diff                        ! norm of difference
   print error
end

end_of_sepran_input
```

Since the right-hand side is a function of x and y we need a subroutine `cfuncf` and hence a main program `helmholtz21`, given by:

```
      program helmholtz21

!     --- Standard main program

      implicit none
      integer, allocatable, dimension (:) :: ibuffr
      integer pbuffr, error
      parameter ( pbuffr=100000000)
      allocate(ibuffr(pbuffr), stat = error)
      if (error /= 0) then
         ! space for these arrays could not be allocated
         print *, "error: (helmholtz21) could not allocate space."
         stop
      end if  ! (error /= 0)
      call sepcombf ( ibuffr, ibuffr, pbuffr )
      end

!     --- Subroutine cfuncf is used to define the right-hand side

      subroutine cfuncf ( ichoice, x, y, z, comval )
      implicit none
      integer ichoice
      double precision x, y, z
      double complex comval
      include 'SPcommon/consta'
      double precision getconst, k
      k = getconst('k')
      if ( ichoice==1 ) then
         comval = -(k**2-5*pi**2)*sin(pi*x)*sin(2d0*pi*y)
      else
         call errchr('cfuncf',1)
         call errsub ( 349, 0, 0, 1)
         call instop
      end if
      end
```

For the shifted Laplace operator we can use the same mesh file and program. The problem files changes only in the matrix input block and the solve input block. Below we give the changed input blocks:

```
matrix, sequence_number = 2
   storage_scheme = compact, complex, shifted_laplace   #  iterative solver
end

!  iteration with shifted Laplace preconditioner with shift = i
```

```
solve, sequence_number = 2  ! use iterative method (bicgstab with ILU precon)
   iteration_method = cg, preconditioner = ilu, print_level = 2, eps = 1d-4 //
      laplace_shift = (0,1)
end
```

Table (3.2.2.1) shows the number of iterations required to solve Equation (3.1.11.2) by a standard Bi-Cgstab method, and for the shifted preconditioner for shifts equal to 0, 1 and i respectively. The results are almost the same as those in Table (3.1.11.1).

Table 3.2.2.1 Number of iterations for several values of the shift

		Bi-CGstab			shift 0			shift 1			shift i		
n		50	100	150	50	100	150	50	100	150	50	100	150
k	10	59	60	84	40	56	84	42	52	86	60	56	84
	20	76	52	56	62	46	48	56	48	42	66	56	58
	30	222	48	32	62	34	30	28	34	26	92	48	38
	40	-	60	26	44	32	24	16	22	20	42	68	22

3.3 Non-linear equations

3.3.1 A special non-linear diffusion equation

This section is under preparation.

3.3.2 The computation of the magnetic field in an alternator

Consider the alternator of Figure 3.3.2.1, consisting of an iron core (region 1) surrounded by vacuum. In regions 2 and 3 a negative and a positive current respectively have been induced. Region 4 is considered separately since this is the region of interest. The magnetic field intensity $\mathbf{H}$ and the

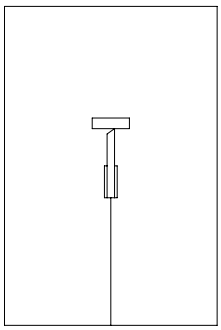


Figure 3.3.2.1: Definition of region for alternator

magnetic flux density $\mathbf{B}$ satisfy the following differential equations (Maxwell equations):

$$-\nabla \times \mathbf{H} = -\mathbf{J} \tag{3.3.2.1}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{3.3.2.2}$$

$$\mathbf{B} = \mu_0 \mu_r \mathbf{H} \tag{3.3.2.3}$$

Both $\mathbf{n} \cdot \mathbf{B}$ and $\mathbf{n} \times \mathbf{H}$ must be continuous over the boundaries.
The electric current density J in regions 2 and 3 are given by:

$$J_2 = -10^8 \ {}^{A}/_{m^2} \tag{3.3.2.4}$$

$$J_3 = 10^8 \ {}^{A}/_{m^2} \tag{3.3.2.5}$$

The relative permeability μ_r in vacuum equals 1. In the iron core we use the non-linear constitutive relations given by Glowinski and Marrocco (1974):

$$\mu_r = \frac{1}{\nu_r(\|B\|)} \qquad \nu_r(\|B\|) = \alpha + (1-\alpha)\frac{\|B\|^8}{\|B\|^8 + \beta} \tag{3.3.2.6}$$

In this example we use $\alpha = 3 \times 10^{-4}$ and $\beta = 16 \times 10^3$. Furthermore $\nu_0 = \frac{1}{\mu_0} = 7.9577471 \times 10^5$.

Since $\mathbf{B}$ is divergence free we can write it as a rotation of a vector potential $\mathbf{A}$.

$$\mathbf{B} = \nabla \times \mathbf{A}, \tag{3.3.2.7}$$

$$\mathbf{H} = \frac{1}{\mu_0 \mu_r} \nabla \times \mathbf{A}. \tag{3.3.2.8}$$

From Equation 3.3.2.1 it follows that

$$-\nabla \times (\frac{1}{\mu_0 \mu_r} \nabla \mathbf{A}) = -\mathbf{J}. \tag{3.3.2.9}$$

Using the vector relation

$$\nabla \times \nabla \times \phi = \nabla(\nabla \cdot \phi) - \nabla^2 \phi, \tag{3.3.2.10}$$

and the notion that $\nabla \cdot \mathbf{A}$ can be chosen zero, we find

$$-\nabla \cdot (\frac{1}{\mu_0 \mu_r} \nabla \mathbf{A}) = \mathbf{J}. \tag{3.3.2.11}$$

In 2D we have $A_x = 0$ and $A_y = 0$, hence

$$-\nabla(\nu_0 \nu_r \nabla A_z) = J_z. \tag{3.3.2.12}$$

At the outer boundary we may use either $A_z = 0$ or $\frac{\partial A_z}{\partial n} = 0$. In this example we use the first option. The continuity at the inner boundaries is automatically satisfied by the finite element method.

In order to get the example into your local directory use the command `sepgetex`:

```
sepgetex magnet
```

To run the example use the commands:

```
sepmesh magnet.msh
seplink magnet
magnet < magnet.prb
seppost magnet.pst
sepview sepplot.001
```

The region is subdivided into triangles by the submesh generators "GENERAL" and "RECTANGLE". As an example linear triangles have been used.
The definition of the curves has been plotted in Figure 3.3.2.2.
SEPMESH needs an input file. An example of an input file for this region is given below:

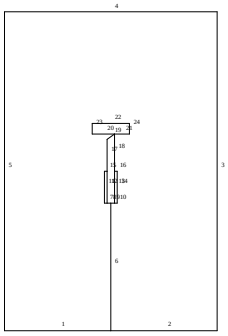


Figure 3.3.2.2: Definition of curves for alternator

```
#  magnet.msh
#
#  mesh file for 2d non-linear magnet problem
#  See Manual Standard Elements Section 3.3.1
#  and Examples manual, Section 3.3.2
```

```
#
#  To run this file use:
#     sepmesh magnet.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
     1: half_width = 0.2       # width of the right part of the outer region
     2: length = 0.6           # length of the outer region
     3: x_core = 0.007         # half_width of the iron core
     4: x_current = 0.012      # at most right x co-ordinate of current region
     5: y_core_low = 0.24      # lower y co-ordinate of the iron core
     6: y_core_upp = 0.36      # upper y co-ordinate of the iron core left
     7: y_current = 0.30       # upper y co-ordinate of current region
     8: y_core_uppr = 0.37     # upper y co-ordinate of the iron core right
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
   coarse(unit=0.01)
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0,3)                        # Lower point at axis
      p2=( half_width,0,3)              # Right under point
      p3=( half_width, length,3)        # Right upper point
      p4=(- half_width, length,3)       # Left upper point
      p5=(- half_width,0,1)             # Left under point
      p6=(0, y_core_low,1)              # lower point of core at axis
      p7=( x_core, y_core_low,0.5)      # lower right point of core
      p8=( x_core, y_core_uppr,1)       # upper right point of core
      p9=( x_core, y_current,0.5)       # upper left point of current region(R)
      p10=( x_current, y_core_low,0.5)  # lower right point of current region(R)
      p11=( x_current, y_current,0.5)   # upper right point of current region(R)
      p12=(- x_core, y_core_upp,1)      # upper left point of core
      p13=(- x_core, y_current,0.5)     # upper right point of current region(L)
      p14=(- x_current, y_current,0.5)  # upper left point of current region(L)
      p15=(- x_current, y_core_low,0.5) # lower left point of current region(L)
      p16=(- x_core, y_core_low,0.5)    # lower left point of core
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Linear elements are used
      c1 = cline (p1,p2)            # Lower boundary right part
      c2 = cline (p2,p3)            # Right-hand side boundary
      c3 = cline (p3,p4)            # Upper boundary
      c4 = cline (p4,p5)            # Left-hand side boundary
      c5 = cline (p5,p1)            # Lower boundary left part
      c6 = cline (p1,p6)            # Lower part of axis
```

```
      c7 = cline (p6,p7)            # Lower right part of iron core
      c8 = curves (c11,c12)         # Right-hand boundary of iron core
      c9 = cline (p12,p8)           # Upper boundary of iron core
      c10= curves (c16,c17)         # Left-hand boundary of iron core
      c11= cline (p7,p9)            # Left-hand boundary of current region(R)
      c12= cline (p9,p8)            # Upper part of right-hand boundary
                                    # of iron core
      c13= cline (p7,p10)           # Lower part of current region(R)
      c14= cline (p10,p11)          # Right-hand boundary of current region (R)
      c15= cline (p11,p9)           # Upper boundary of current region (R)
      c16= cline (p12,p13)          # Upper part of left-hand boundary
                                    # of iron core
      c17= cline (p13,p16)          # Right-hand boundary of current region(L)
      c18= cline (p16,p15)          # Lower part of current region(L)
      c19= cline (p15,p14)          # Left-hand boundary of current region (L)
      c20= cline (p14,p13)          # Upper boundary of current region (L)
      c21= cline (p6,p16)           # Lower left part of iron core
      c22= curves(c1,c2,c3,c4,c5)         # Outer boundary
      c23= curves(c21,c18,c19,c20,-c16,c9,-c12,-c15,-c14,-c13,-c7)
      c24= curves(-c21,c7)
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
                    # Linear triangles are used
      s1=general3(c24,c8,-c9,c10)          # iron core
      s2=rectangle3(c13,c14,c15,-c11)      # current region (R)
      s3=rectangle3(c18,c19,c20,c17)       # current region (L)
      s4=general3(c22,c6,c23,-c6)          # vacuum

#
#  Connect elements groups to surfaces
#
   meshsurf         # See Users Manual Section 2.4
      selm1 = s1
      selm2 = s2
      selm3 = s3
      selm4 = s4

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2

end
```

Figure 3.3.2.3 shows the mesh generated by SEPMESH.

In order to solve this problem we need to use elements of type 800 for the vacuum and of type 803 for the iron core.
The non-linear problem is solved by a Newton linearization method. Unfortunately this method does not converge for a current density of $10^{8A}/_{m^2}$ For that reason we start with a current density of $10^{7A}/_{m^2}$ and gradually increase this value by steps of $2 \times 10^{7A}/_{m^2}$ until the final value has been reached. Such a method, in which a significant parameter is changed gradually, is called a continuation method.
The input for the continuation method can be found in the input block "NONLINEAR_EQUATIONS". The minimum number of iterations is set to 6, in order to ensure that the final value will be reached. As start vector for the iteration the zero vector is used.

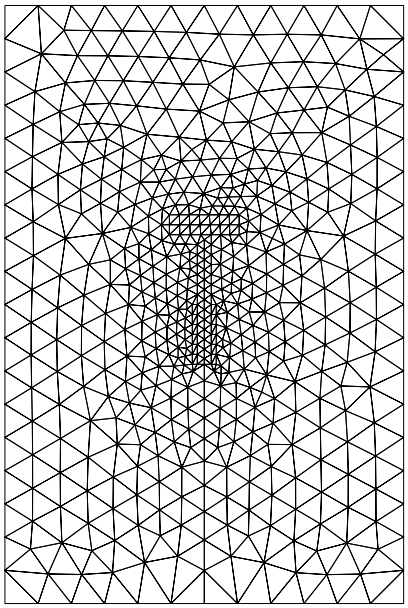


Figure 3.3.2.3: Plot of mesh generated by SEPMESH

An important quantity to be computed is of course the magnetic flux density $\mathbf{B}$, which is done by computing the gradient of the potential. Another important issue is the magnitude of $\mathbf{B}$. In order to compute all these quantities it is necessary to introduce the input block "STRUCTURE", which defines how the process develops.

- First the initial vector is created.
- Next the non-linear equation is solved and A_z is stored in vector V_1.
- Then $\mathbf{B}$ is computed as a derivative and stored in vector V_2.
- Finally the magnitude $\|\mathbf{B}\|$ is computed and stored in vector V_3.
- Although superfluous, the input block "STRUCTURE" is closed by the *output* command, which writes all three vectors.

Since we have to add the unction subroutine $funcc2$, in order to define μ and $\frac{\mu}{B}$ it is not possible to use program SEPCOMP, but we have to write out main program *magnet*, which consists of three statements only. The listing for this program is given by:

```
program magnet
```

```
!     main program for 2d non-linear magnet problem
!     See Manual Standard Elements Section 3.3.1
!     and Examples manual, Section 3.3.2
!
!     To link this file use:
!     seplink magnet

      call sepcom(0)

      end

!     --- subroutine funcc2 is used to define nu and d nu / dB
!         as function of the B computed before

      subroutine funcc2 ( ichois, x, y, z, graphi, alpha, dalpha )
      integer ichois
      double precision x, y, z, graphi, alpha, dalpha

!     The formula for nu(|b|) can be found in
!     R. Glowinski and A. Marocco
!     Analyse numerique du champ magnetique d'un alternateur par
!     element finis et sur-relaxation ponctuelle non lineaire
!     Computer Methods in applied mechanics and engineering 3 (1974), 55-85

      double precision co1, co2, anu, fac, grap

      co1    = 3d-4
      co2    = 1.6d4
      anu    = 7.9577471d5
      grap   = abs(graphi)
      fac    = grap**8
      alpha  = anu*(co1+(1d0-co1)*fac/(fac+co2))
      dalpha = 8d0*anu*(1d0-co1)*co2*(grap**7)/((fac+co2)*(fac+co2))
      end
```

This program needs an input file which is the same as for SEPCOMP. The following input file may be used to solve the problem:

```
# magnet.prb
#
#  problem file for 2d non-linear magnet problem
#  See Manual Standard Elements Section 3.3.1
#  and Examples manual, Section 3.3.2
#
#  To run this file use:
#     magnet < magnet.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
```

```
   reals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
   vector_names
      potential
      magnetic_field_strength
      magnitude_of_magnetic_field
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1,(type=803)        # Type number for non-linear diffusion equation
                               # iron core
      elgrp2,(type=800)        # Type number for linear diffusion equation
      elgrp3,(type=800)
      elgrp4,(type=800)
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
                               # Only velocities are prescribed, not the
                               # pressures
      curves (c1 to c5)        # The potential is prescribed on the outer
                               # boundary, curves c1 to c5
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   storage_scheme = compact, symmetric # Symmetrical compact matrix
                             # So an iterative method will be applied
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because the integral of the pressure over the boundary
#  is required

structure                   # See Users Manual Section 3.2.3
   # create the start vector for the non-linear iteration
   create_vector, potential  # make vector 0

   # compute the potential by solving a system of non-linear equations
   solve_nonlinear_system, potential

   # compute the magnetic field strength by computing the gradient of the
   # potential
   derivatives, magnetic_field_strength

   # compute the magnitude of the magnetic field strength
!   compute_vector  magnitude_of_magnetic_field//
```

```
!       length vector  magnetic_field_strength
   magnitude_of_magnetic_field = length vector  magnetic_field_strength

end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations
   global_options, maxiter=15, accuracy = 1d-4, miniter=6, print_level=2
   equation 1
      fill_coefficients = 1
      change_coefficients
         at_iteration 2, sequence_number = 1
         at_iteration 3, sequence_number = 2
         at_iteration 4, sequence_number = 3
         at_iteration 5, sequence_number = 4
end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1(nparm=20)              # iron-core (type=803)
      icoef5 = 2                 # Picard iteration
   elgrp2(nparm=20)              # source with negative current
      coef6 = 7.9577471d5        # nu = 1/(4d-7*pi)
      coef9 = coef6
      coef16= -2d7               # f at start = -2d7
   elgrp3(nparm=20)              # source with positive current
      coef6 = 7.9577471d5        # nu = 1/(4d-7*pi)
      coef9 = coef6
      coef16= 2d7                # f at start = 2d7
   elgrp4(nparm=20)              # vacuum, no source
      coef6 = 7.9577471d5        # nu = 1/(4d-7*pi)
      coef9 = coef6
      coef16= 0                  # f=0
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number=1
   elgrp2
      coef16=-4d7               # f at second iteration is -4d7
   elgrp3
      coef16= 4d7               # f at second iteration is 4d7
end
change coefficients, sequence_number=2
   elgrp2
      coef16=-6d7               # f at third iteration is -6d7
   elgrp3
      coef16= 6d7               # f at third iteration is 6d7
end
```

```
change coefficients, sequence_number=3
   elgrp2
      coef16=-8d7              # f at fourth iteration is -8d7
   elgrp3
      coef16= 8d7              # f at fourth iteration is 8d7
end
change coefficients, sequence_number=4
   elgrp2
      coef16=-1d8              # f at other iterations is -1d8
   elgrp3
      coef16= 1d8              # f at other iterations is 1d8
end

# input for linear solver
# See Users Manual Section 3.2.8

solve
   iteration_method=cg,accuracy = 1d-5,print_level=0
end

# input for derivatives, i.e. computation of the magnetic field strength
# See Users Manual Section 3.2.11 and Standard Problems Section 3.1

derivatives
   icheld = 2                    # Compute gradient
   element_groups = 4            # The magnetic field strength is only computed
                                 # in the outer field
end

end_of_sepran_input
```

Once the solution has been computed, it may be printed and plotted by the postprocessing program SEPPOST. SEPPOST also requires an input file. Mark that the numbering of vectors in SEPCOMP and SEPPOST differ by one. hence now V_1 is called V_0 and so on.
The following input file plots the curves of the region, with and without curve numbers, plots the mesh and a part of the mesh, plots the equi-potential lines with two different types of levels as well as restricted to region 4 and finally plots the vector $\mathbf{B}$ and its magnitude in region 4.

```
# magnet.pst
#
#  Input file for postprocessing for 2d non-linear magnet problem
#  See Manual Standard Elements Section 3.3.1
#  and Examples manual, Section 3.3.2
#
#
#  To run this file use:
#     seppost magnet.pst > magnet.out
#
#  Reads the files meshoutput and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2

  # Plot the complete mesh

   plot mesh
```

```
  # Plot a part of the mesh in the outer region

   plot mesh, skip element groups(1,2,3), region=(-0.035,0.035,0.37,0.39)

  # Make contour plots of the potential

   plot contour potential
   plot contour potential,minlevel=-0.02, maxlevel=0.02,nlevel=21
   plot contour potential,minlevel=-0.0035, maxlevel=0.0035,nlevel=21//
    region=(-0.035,0.035,0.37,0.39)

  # Make a contour plot of the magnetic field strength

   plot vector magnetic_field_strength, region=(-0.06,0.06,0.35,0.41)

  # Make a contour plot of the magnitude of the magnetic field strength

   plot contour magnitude_of_magnetic_field, region=(-0.06,0.06,0.35,0.41)

end
```

Figure 3.3.2.4 shows the contour plots made by program SEPPOST. Figure 3.3.2.5 shows the vector plot and the contour plot of the magnitude of **B** in region 4. These plots may be visualized by the program SEPDISPLAY or SEPVIEW.

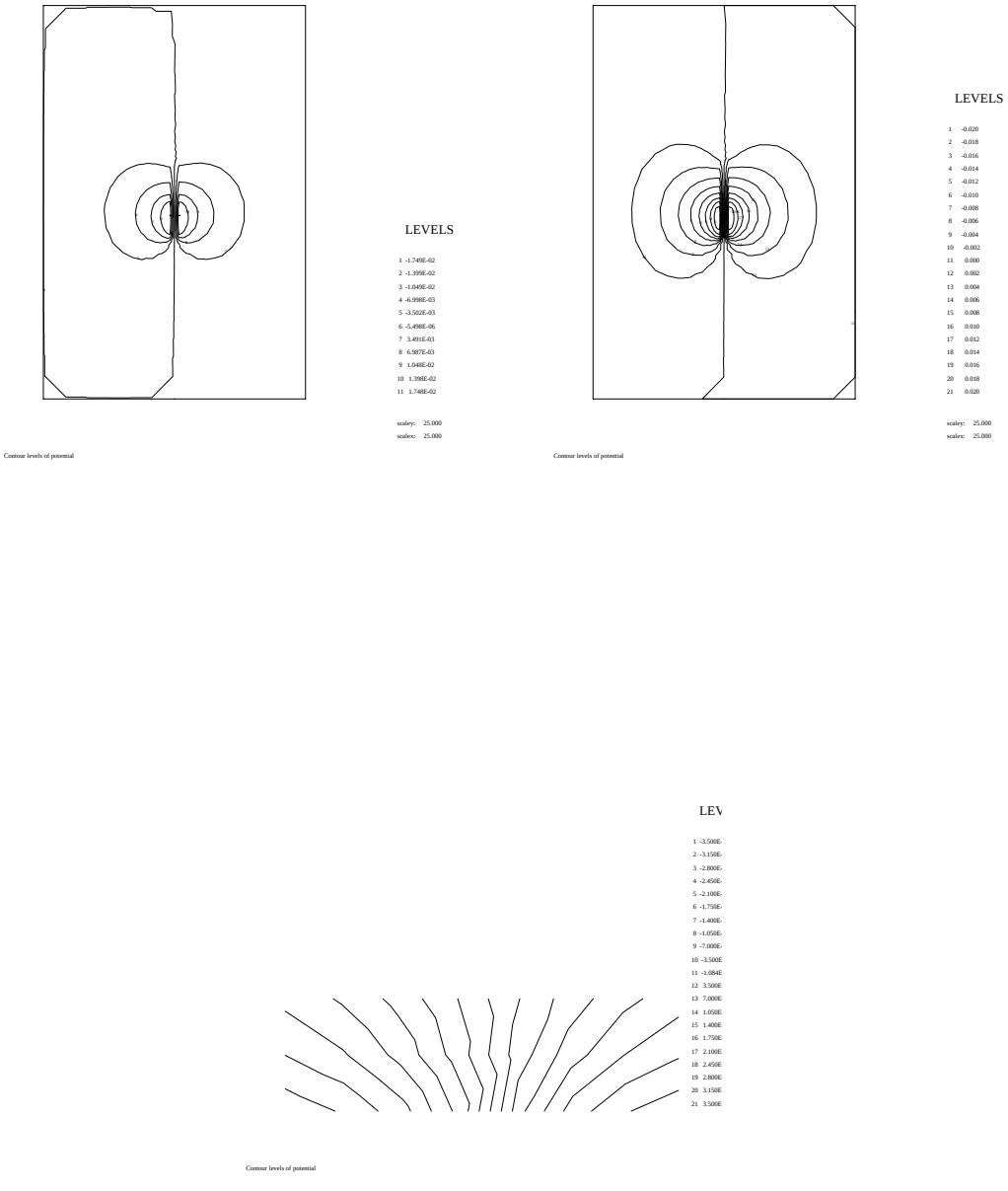


Figure 3.3.2.4: Contour plots generated by SEPPOST

3.3.3 The solution of Hamilton-Jacobi-Bellman equation

In this section we consider special Hamilton-Jacobi-Bellman differential equation:

$$-\epsilon\Delta c + |\frac{\partial c}{\partial x}| + |\frac{\partial c}{\partial y}| = 1 \qquad (3.3.3.1)$$

This is a typical non-linear equation due to the modulus of the convective terms. Equation 3.3.3.1 is solved on the square (-1,-1) $\times$ (1,1) and provided with the Dirichlet boundary condition $c = 0$ at the complete boundary.
Since the equation is non-linear an iteration procedure to solve it is necessary. We start for example by the solution of the Poisson equation (no convective terms) and use this solution in the next iteration to compute the sign of the derivatives. This procedure is repeated until convergence is reached.
For small values of ε it is known that the first derivatives of the solution is discontinuous in the neighbourhood of the lines $x = y$ and $x = -y$. For that reason the mesh is adapted to these lines.
The following input file may be used to generate the mesh by program sepmesh:

```
* modconv2.msh
mesh2d
   coarse(unit=0.03)
   points
      p1=(-1,-1)
      p2=(1,-1)
      p3=(1,1)
      p4=(-1,1)
      p5=(0,0)
   curves
      c1=cline1(p1,p2)
      c2=cline1(p2,p3)
      c3=cline1(p3,p4)
      c4=cline1(p4,p1)
      c5=cline1(p1,p5)
      c6=cline1(p2,p5)
      c7=cline1(p3,p5)
      c8=cline1(p4,p5)
   surfaces
      s1=general3(c1,c6,-c5)
      s2=general3(c2,c7,-c6)
      s3=general3(c3,c8,-c7)
      s4=general3(c4,c5,-c8)
   plot
end
```

Figure 3.3.3.1 shows the mesh generated by SEPMESH.

In order to solve this problem we need to use elements of type 800. We start with the solution of the Poisson equation, without convective terms. Next we proceed with the special equation, which requires that the integer coefficient 5 is set equal to 2.
The system of linear equations that arises in each step of the non-linear iteration process is solved by an iterative solver (CGSTAB) using the solution of the previous iteration as a start.
Experiments showed that for convergence it was necessary to solve these equations rather accurate ($\epsilon = 10^{-4}$) and for small values of ε it was necessary to use upwinding in order to get convergence and an accurate solution.
The following input file may be used to solve the problem by program sepcomp in the case $\varepsilon = 10^{-3}$:

```
* modconv2.prb
```

```
constants
   vector_names
      solution
end
problem
   types
      elgrp1=(type=800)           # Use standard type 800
   essbouncond
      curves(c1,c4)               # essential boundary conditions on each side
end
matrix
   storage_method = compact     # compact storage for iterative solution
end
coefficients, sequence_number=1
   elgrp1(nparm=20)               #  Poisson equation (first step)
      coef6 = 0.001               #  eps
      coef9 = coef 6              #  eps
      coef16= 1                   #  f
end
change coefficients, sequence_number=1
   elgrp1(nparm=20)         #  Special equation (next steps)
      icoef2 = 3            #  upwind
      icoef5 = 2            #  absolute values of convective terms
end
create vector
end
nonlinear_equations
   global_options, accuracy=1d-2, print_level=2, maxiter=10//
   at_error = return, lin_solver=1
   equation 1
      fill_coefficients = 1                  # start with Poisson
   change_coefficients
      at_iteration 2, sequence_number=1      # resume with special equation
end
solve, sequence_number=1
   iteration_method=cg, accuracy=1d-4, start=old_solution, print_level=1
end
end_of_sepran_input
```

Once the solution has been computed, it may be printed and plotted by the postprocessing program SEPPOST. SEPPOST also requires an input file. Mark that the numbering of vectors in SEPCOMP and SEPPOST differ by one. hence now V_1 is called V_0 and so on.
The following input file makes a 3D plot of the solution as well as a contour plot.

```
* modconv2.pst
postprocessing
   3d plot solution
   plot contour solution
end
```

Figure 3.3.3.2 shows the 3D plot made by program SEPPOST and Figure 3.3.3.3 shows the contour plot.

3.3.4 An example of non-linear convection

In this section we consider a mathematical example of non-linear convection as treated in Section 3.1. The example is strongly related to the example of Section 3.3.3.

$$-\epsilon\Delta c + (\frac{\partial c}{\partial x})^2 + (\frac{\partial c}{\partial y})^2 = 1 \tag{3.3.4.1}$$

This is a typical non-linear equation due to the quadratic convective terms. Equation 3.3.4.1 is solved on the square (-1,-1) $\times$ (1,1) and provided with the Dirichlet boundary condition $c = 0$ at the complete boundary.
Since the equation is non-linear an iteration procedure to solve it is necessary. We start for example by the solution of the Poisson equation (no convective terms) and use this solution in the next iteration to compute the sign of the derivatives. This procedure is repeated until convergence is reached.
For small values of ε it is known that the first derivatives of the solution is discontinuous in the neighbourhood of the lines $x = y$ and $x = -y$. For that reason the mesh is adapted to these lines. The following input file may be used to generate the mesh by program sepmesh:

```
* convnon2.msh
mesh2d
   coarse(unit=0.10)
   points
      p1=(-1,-1)
      p2=(1,-1)
      p3=(1,1)
      p4=(-1,1)
      p5=(0,0)
   curves
      c1=cline1(p1,p2)
      c2=cline1(p2,p3)
      c3=cline1(p3,p4)
      c4=cline1(p4,p1)
      c5=cline1(p1,p5)
      c6=cline1(p2,p5)
      c7=cline1(p3,p5)
      c8=cline1(p4,p5)
   surfaces
      s1=general3(c1,c6,-c5)
      s2=general3(c2,c7,-c6)
      s3=general3(c3,c8,-c7)
      s4=general3(c4,c5,-c8)
   plot
end
```

Mark that this mesh is identical to the one in Section 3.3.3, however, with a large coarseness and hence less elements.

In order to solve this problem we need to use elements of type 800. We start with the solution of the Poisson equation, without convective terms. Next we proceed with the non-linear convection, which requires that the integer coefficient 5 is set equal to 3.
This problem requires a user written subroutine FUNCC2 in which the function of the gradient must be evaluated as well as its partial derivatives with respect to the gradient of the solution. Hence we get:

$$g(\mathbf{x}) = (\frac{\partial c}{\partial x})^2 + (\frac{\partial c}{\partial y})^2 \tag{3.3.4.2}$$

$$(\frac{\partial g}{\partial \nabla c}) = 2 \begin{pmatrix} \frac{\partial c}{\partial x} \\ \frac{\partial c}{\partial y} \end{pmatrix} \tag{3.3.4.3}$$

The system of linear equations that arises in each step of the non-linear iteration process is solved by an iterative solver (CGSTAB) using the solution of the previous iteration as a start.
Experiments showed that for convergence it was necessary to solve these equations rather accurate ($\epsilon = 10^{-4}$) and for small values of ε it was necessary to use upwinding in order to get convergence and an accurate solution.
Since a user written subroutine is provided, it is also necessary to create your main program. This program consists only of a call to subroutine sepcom.

```
      program convnon2
      call sepcom ( 0 )
      end

      subroutine funcc2 ( ichoice, x, y, z, gradc, g, dgdgrad )
      implicit none
      integer ichoice
      double precision x, y, z, gradc(*), g, dgdgrad(*)
      g = gradc(1)**2 + gradc(2)**2
      dgdgrad(1) = 2d0*gradc(1)
      dgdgrad(2) = 2d0*gradc(2)
      end
```

The following input program may be used to solve the problem The corresponding input file in the case $\varepsilon = 10^{-3}$ is:

```
* convnon2.prb
problem
   types
      elgrp1=(type=800)           # Use standard type 800
   essbouncond
      curves(c1,c4)               # essential boundary conditions on each side
end
matrix
   method=6                       # compact storage for iterative solution
end
coefficients, sequence_number=1
   elgrp1(nparm=20)               #  Poisson equation (first step)
      coef6 = 0.001               #  eps
      coef9 = coef 6              #  eps
      coef16= 1                   #  f
end
change coefficients, sequence_number=1
   elgrp1(nparm=20)          #  Special equation (next steps)
      icoef2 = 3             #  upwind
      icoef5 = 3             #  Nonlinear convective terms with newton
end
create vector
end
nonlinear_equations
   global_options, accuracy=1d-2, print_level=2, maxiter=20//
   at_error = return, lin_solver=1, criterion = relative
   equation 1
```

```
      fill_coefficients = 1                  # start with Poisson
   change_coefficients
      at_iteration 2, sequence_number=1      # resume with special equation
end
solve, sequence_number=1
   iteration_method=cg, accuracy=1d-4, start=old_solution, print_level=2
end
end_of_sepran_input
```

In order to run this program we have to link it by seplink and than run it.
Hence:

```
seplink convnon2
convnon2 < convnon2.prb > convnon2.out
```

Once the solution has been computed, it may be printed and plotted by the postprocessing program SEPPOST. SEPPOST also requires an input file. Mark that the numbering of vectors in SEPCOMP and SEPPOST differ by one. hence now V_1 is called V_0 and so on.
The following input file makes a 3D plot of the solution as well as a contour plot.

```
* convnon2.pst
postprocessing
   name v0 = solution
   3d plot v0
   plot contour v0
end
```

Figure 3.3.4.1 shows the 3D plot made by program SEPPOST and Figure 3.3.4.2 shows the contour plot.

3.3.5 An example of compressible potential flow

In this section we consider a compressible flow through a nozzle as sketched in Figure 3.3.5.1.
In order to get this example in your local directory use the command

```
sepgetex nozzle
```

To run the example use the following commands:

```
seplink nozzlemesh
nozzlemesh < nozzle.msh
view mesh
seplink nozzle
nozzle < nozzle.prb
seppost nozzle.pst
view results
```

If we assume that the flow is stationary, frictionless and isentropic, then the flow can be considered as a potential flow.

Let $\boldsymbol{u}$ be the velocity of the fluid, p the pressure and ρ the density.
Define the potential φ such that $\boldsymbol{u} = \nabla\varphi$.
From the continuity equation it follows that

$$-div(\rho\nabla\varphi) = 0 \tag{3.3.5.1}$$

In case of an incompressible flow the density ρ is constant. For a compressible flow the density depends on the equation of state. For a ideal gas the following relation can be derived of the various equations.

$$\rho = \rho_0(1 - \frac{\gamma-1}{\gamma+1}\frac{1}{C_*^2}\|\nabla\varphi\|^2)^{\frac{1}{\gamma-1}}$$

with γ the ratio of specific heats ($\gamma = 1.4$ in air),
ρ_0 the density for U_∞ (the velocity at $x = \infty$),
C_* the velocity of sound.
With $\|\nabla\varphi\|^2$ we mean the Euclidean norm:

$$\|\nabla\varphi\|^2 = (\frac{\partial\varphi}{\partial x})^2 + (\frac{\partial\varphi}{\partial y})^2$$

Boundary conditions with respect to the problem:

We assume that the wall of the nozzle is impermeable, hence $\boldsymbol{u}\cdot\boldsymbol{n} = 0$

We assume that at the inflow and outflow the flow is one-dimensional with size U_∞, hence:

$$\boldsymbol{u}|_{x=-L} = \boldsymbol{u}|_{x=L} = \begin{pmatrix} U_\infty \\ 0 \end{pmatrix}$$

Because of the symmetry it is sufficient to consider only the top half of the nozzle.
The curve that defines the top wall of the nozzle can be approximated by the following formula:

$$\begin{aligned} -10 \le x \le -5 \qquad & y(x) = 1 \\ -5 < x < 5 \qquad & y(x) = 1 - \alpha e^{(-\beta x^2)} \\ 5 \le x \le 10 \qquad & y(x) = 1 \end{aligned}$$

Program `sepmesh` may be used to create a mesh for this problem. Since the upper wall is given by a function we need a user written function funccv. Therefore program `sepmesh` is replaced by a program `nozzlemesh` that contains the subroutine `funccv`:

```
      program nozzlemesh

!     --- Main program to create the mesh for the nozzle in example 3.3.5
!         of the Examples Manual
!         This main program is necessary in order to provide a function funccv
!         which defines the parameter curve

      call sepmsh ( 0 )
      end


      subroutine funccv ( icurve, t, x, y, z )

!     --- Function subroutine to define the upper wall in the nozzle of
!         example 3.3.5 of the Examples Manual
!         The curve is defined as follows:
!         x < -5: y(x) = 1
!         -5 <= x <= 5:  y(x) = 1- alpha exp(-beta x^2)
!         x > 5: y(x) = 1

      implicit none
      integer icurve
      double precision t, x, y, z
      double precision alpha, beta
      double precision getconst

!     --- alpha and beta are provided as real constants in the input file
!         x is equal to the parameter t

      alpha = getconst('alpha')
      beta  = getconst('beta')
      x = t
      if ( abs(t)>5 ) then
         y = 1d0
      else
         y = 1d0 - alpha*exp(-beta*x**2)
      end if
      end
```

This program requires input form the input file:

```
#  nozzle.msh
#
#  mesh file for 2d nozzle problem
#  See Examples Manual Section 3.3.5
#
#  To run this file use:
#     sepmesh nozzle.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
```

```
     half_width = 1         # width of the upper half of the channel
     half_length = 10       # half the length of the channel
     alpha = 0.7            # parameter defining the nozzle
                            # the half width at the smallest part is
                            # 1-alpha
     beta = 0.4             # parameter defining the nozzle
   integers
      n = 50           # number of elements in length direction
      m = 6            # number of elements in width direction
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=( -half_length,0)              # Left under point
      p2=( half_length,0)               # Right under point
      p3=( half_length, half_width)     # Right upper point
      p4=( -half_length, half_width)    # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Linear elements are used
      c1=line  (p1,p2,nelm= n)     # symmetry axis
      c2=line  (p2,p3,nelm= m)     # outflow boundary
      c3=param (p3,p4,nelm= n,init= half_length,end=- half_length)
                                   # upper wall is defined by a parameter
                                   # function, with t=x
      c4=line  (p4,p1,nelm= m) # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
                    # Linear triangles are used
      s1=rectangle3(c1,c2,c3,c4)

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2

end
```

Figure 3.3.5.2 shows the mesh created by nozzlemesh.

Equation 3.3.5.1 is non-linear because ρ depends on the potential φ and therefore this equation can be considered as a special case of Equation (3.3.1.1) in the manual STANDARD PROBLEMS. This implies that the user must provide a function subroutine `funcc2`, which defines ρ and $\frac{\partial \rho}{\partial \varphi}$ as function of φ. See Section 3.3.1.

This subroutine is used in the following file `nozzle.f`

```
      program nozzle

!     --- Main program to solve the non-linear potential problem in the nozzle
```

```
!         example 3.3.5 of the manual Examples
!         This main program is necessary in order to provide a function funcc2
!         which defines the dependence of the density on the gradient of the
!         potential phi

      call sepcom ( 0 )
      end


      subroutine funcc2 ( ichoice, x, y, z, gradc, g, dgdgrad )

!     --- Function subroutine to define the density in the non-linear potential
!         problem in the nozzle as function of the gradient of the potential
!         In case of Newton also the derivative with respect to
!         the norm of the gradient is defined
!         example 3.3.5 of the manual Standard Problems
!         The density for the compressible potential flow is defined by
!
!         rho = rho_0 c2 ^ 1/(gamma-1)
!         with
!         c2 = 1 - (gamma-1)/(gamma+1) (1/C*)^2 ||grad phi||^2
!
!         The derivative d rho / d ||grad phi|| by
!
!         d rho / d ||grad phi|| = rho_0/(gamma-1) c2^(1/(gamma-1)-1) *
!              (-2 (gamma-1)/(gamma+1) (1/C*)^2 ||grad phi|| )

      implicit none
      integer ichoice
      double precision x, y, z, gradc, g, dgdgrad
      double precision rho_0, cstar, gamma, gmin1, gplus1, c, c1, c2
      double precision getconst

!     --- rho_0, gamma and C* are provided as real constants in the input file

      rho_0 = getconst('rho_0')
      gamma = getconst('gamma')
      cstar = getconst('Cstar')
      gmin1 = gamma-1d0
      gplus1 = gamma+1d0
      c = gmin1/gplus1
      c1 = c/cstar**2
      c2 = 1d0-c1*gradc**2

!     --- The function rho is stored in g

      g = rho_0*c2**(1d0/gmin1)

      if ( ichoice==2 ) then

!     --- ichoice = 2, Newton linearization;
!         also  d rho / d ||grad phi|| is required

         dgdgrad = rho_0/gmin1*c2**(1d0/gmin1-1d0)*(-2d0*c1*gradc)
```

```
      end if
      end
```

The corresponding input file is given by

```
# nozzle.prb
#
#  problem file for 2d nozzle problem
#  non-linear potential flow problem
#  See Examples Manual Section 3.3.5
#
#  To run this file use:
#     sepcomp nozzle.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      rho_0 = 1.2           # density where u = U_inf
      gamma = 1.4           # specific heat ratio (air)
      Cstar = 340           # Velocity of sound
      u_infinity = 50       # Velocity at infinity U_inf
      m_infinity = 60       # Momentum at infinity rho_0 U_inf
   vector_names
      potential             # unknown phi
      velocity              # derived quantity u = grad phi
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=803                # Type number for non-linear diffusion equation
                                # See Standard problems Section 3.1
                                # Define the type of natural boundary conditions
   natbouncond
      bngrp1 = type=801         # Boundary group 1, standard natural boundary
                                # condition for diffusion equation
      bngrp2 = type=801         # Boundary group 2, standard natural boundary
                                # condition for diffusion equation
   bounelements                 # Defines where natural boundary conditions
                                # are given
      belm1 = curves(c2)        # On curve 2: boundary group 1
      belm2 = curves(c4)        # On curve 4: boundary group 2
end
#
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
```

```
#  This is necessary because the integral of the pressure over the boundary
#  is required
#
structure                   # See Users Manual Section 3.2.3

   # Create start vector for the potential

   create_vector potential

   # Compute the potential, by solving the non-linear equations

   solve_nonlinear_system, potential

   # Compute the velocity as gradient of the potential

   derivatives,velocity

   # Write the results to sepcomp.out

   output

end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   storage_scheme=compact, symmetric  # Symmetrical compact matrix
                               # So an iterative method will be applied
end

# Create start vector
#  See Users Manual Section 3.2.5

create
# The start vector is set equal to zero, so no extra information is required
end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.3

coefficients
   elgrp1 ( nparm=20 )     # The coefficients for the non-linear diffusion
                           # equation are defined by 20 parameters
      icoef5 = 1           # 5:  Type of linearization (1=Picard)
   bngrp1 ( nparm=15)      # The natural boundary condition requires
                           # 15 parameters
      coef7 =  m_infinity  # On c2 we have alpha d phi / d n = m_inf
   bngrp2 ( nparm=15)
      coef7 = - m_infinity # On c4 we have alpha d phi / d n = - m_inf
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7
```

```
change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations
   global_options, maxiter=20, accuracy=1d-4,print_level=2, lin_solver=1//
   at_error return
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 1
end

# compute velocity
# See Users Manual, Section 3.2.11

derivatives
   icheld=2          # icheld=2, velocity in nodes
                     # See Standard problems Section 3.1
end

# input for the linear solver
# See Users Manual, Section 3.2.8

solve
   iteration_method = cg, accuracy=1d-2//
   termination_crit = rel_residual, start=old_solution//
   print_level=0
end

end_of_sepran_input
```

In order to view the computed potential and the corresponding velocity program seppost may be used with the following input:

```
#  nozzle.pst
#  Input file for postprocessing for 2d nozzle problem
#  non-linear potential flow problem
#  See Examples Manual Section 3.3.5
#
#
#  To run this file use:
#     seppost nozzle.pst > nozzle.out
#
```

```
#  Reads the files meshoutput, sepcomp.inf and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2

#  Plot the results
#  See Users Manual Section 5.4

   plot vector velocity              # Vector plot of velocity
   plot contour potential            # Contour plot of pressure
   plot coloured contour potential

end
```

Figure 3.3.5.3 shows the computed velocities, and Figures 3.3.5.4 and 3.3.5.5 the contourlines and colored contour levels of the potential

3.4 δ-type source terms

This section is under preparation

3.5 Second order real linear elliptic and parabolic equations with two degrees of freedom

3.5.1 Falling film absorption with a large heat effect in one-dimensional film flow

Falling film absorption accompanied by a large heat effect is encountered in absorption heat pumps or cooling machines and in some industrial applications like the absorption of ammonia or hydrochloric acid as well as in strongly exothermic reactions like detergent making by sulfonation of organic alkylates (Yih, 1986). Figure 3.5.1.1 gives the geometry of the absorber. On one side of the plate a solution of substance 1 in substance 2 flows down as a thin laminar film. At the liquid-vapour interface the vapour (substance 1) is absorbed and then transported into the bulk of the film. The heat of absorption is released at the interface and transported through the film and the wall to the cooling medium. The cooling medium flows on the other side of the plate parallel to the film (cocurrent or countercurrent flow), or in a direction perpendicular to the plane of illustration (cross-flow).
In case of a steady state, constant properties, a film thickness and velocity that are not influenced by the vapour absorption, and only diffusion of heat and mass perpendicular to the wall and convection along the wall, the absorption in the liquid film is described by the following dimensionless convection diffusion equations (van der Wekken and Wassenaar, 1988).

$$U\frac{\partial\gamma}{\partial Gz} = Le\frac{\partial^2\gamma}{\partial Y^2} \tag{3.5.1.1}$$

$$U\frac{\partial\theta}{\partial Gz} = \frac{\partial^2\theta}{\partial Y^2} \tag{3.5.1.2}$$

Here U denotes the velocity along the wall, normalized on the average velocity,
Gz is the co-ordinate along the wall, normalized on a characteristic length for heat transfer (Graetz number),
Y is the co-ordinate perpendicular to the wall, normalized on the film thickness,
γ is the normalized mass fraction of the volatile component,
θ is the normalized temperature, and
Le is the Lewis number, the ratio between mass and heat diffusivity.
A two-dimensional version of this problem is considered in van der Wekken et al. (1988).

The mixture enters the absorber at $Gz = 0$ at uniform temperature and mass fraction, the wall $Y = 1$ is impermeable for mass, but there is a cooling condition for heat ($Bi \to \infty$: isothermal wall, $Bi = 0$: adiabatic wall). At the interface $Y = 0$ there is thermodynamic equilibrium between vapour and liquid, and the heat released is proportional to the absorbed mass (Equation 3.5.1.4). In dimensionless form the boundary conditions transform to:

$$Gz = 0; 0 \le Y \le 1 : \theta = 0, \gamma = 0 \tag{3.5.1.3}$$

$$Gz > 0; Y = 1 : \theta + \gamma = 1, \frac{\partial\theta}{\partial Y} = \Lambda\frac{\partial\gamma}{\partial Y} \tag{3.5.1.4}$$

$$Gz > 0; Y = 0 : \frac{\partial\gamma}{\partial Y} = 0, \frac{\partial\theta}{\partial Y} = Bi(\theta - \theta_c) \tag{3.5.1.5}$$

Here Λ is the dimensionless heat of absorption, Bi is the Biot number, the ratio of heat transfer in the film to that to the cooling medium, θ_c is the dimensionless cooling medium temperature. A version of this problem with co- or countercurrent flow cooling is elaborated in Wassenaar (1994).

The boundary conditions 3.5.1.4 at the phase change surface $Y = 1$ are similar to the conditions at $x = s(t)$ in the solidification problem in Section 6.1. The difference is that in the above case there are two components in the densest phase. The Gibbs phase rule then dictates that there is still one thermodynamic degree of freedom, so that if the pressure p is fixed, the temperature is not fixed, but still depends on the mass fraction, a relation that is linearized in 3.5.1.4.

The region is subdivided into triangles by the submesh generator "RECTANGLE". As an example linear triangles have been used.
SEPMESH needs an input file. An example of an input file for this region is given below:

```
#  absorb.msh
#
# mesh for absorber
#
#   P5           c4          P4
#     *----------<---------*
#  c5 ^                     ^ c3
#   P6*                     *P3
#     |      S1             |
#  c6 ^                     ^ c2
#     |                     |
#     *---------->---------*
#   P1           c1          P2
#
#
mesh2d
   points
      p1=(0,0)
      p2=(1000,0)
      p3=(1000,0.9)
      p4=(1000,1.0)
      p5=(0,1.0)
      p6=(0,0.9)
   curves
      c1=line1(p1,p2,nelm=60,ratio=2,factor=1.17707)
      c2=line1(p2,p3,nelm=15)
      c3=line1(p3,p4,nelm=15,ratio=3,factor=40)
      c4=translate c1(p5,p4)
      c5=translate c3(p6,p5)
      c6=translate c2(p1,p6)
      c7=curves(c2,c3)
      c8=curves(c6,c5)
   surfaces
      s1=rectangle3(c1,c7,-c4,-c8)
   plot
end
```

In order to solve the problem program SEPCOMP is used. The differential equation as well as most of the boundary conditions are standard and do not need any special explanation. The boundary conditions at the side $Gz > 0; Y = 1$, however, are special since they contain linear combinations of γ and θ. In order to be able to deal with these boundary conditions it is necessary to use local transformations.
Let us define the vectors $\mathbf{T}$ and $\tilde{\mathbf{T}}$ by:

$$\mathbf{T} = \begin{pmatrix} \theta \\ \gamma \end{pmatrix}, \quad \tilde{\mathbf{T}} = \begin{pmatrix} \theta \\ \theta + \gamma \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \mathbf{T} \tag{3.5.1.6}$$

In order to satisfy the boundary condition $\theta + \gamma = 1$ at the boundary $Gz > 0; Y = 1$, we transform the unknowns at that boundary from $\mathbf{T}$ to $\tilde{\mathbf{T}}$ using the transformation given in Equation 3.5.1.6. After this transformation the second unknown at the boundary is prescribed (essential boundary condition) and has value 1.
The transformation is defined in the input of program SEPCOMP in the input block PROBLEM

under the keyword `localtransform`. The transformation matrix is defined in matrixr.
In order to satisfy the other boundary condition at $Gz > 0; Y = 1$, we follow the method in van der Wekken et al (1988). This method requires some knowledge of weak formulations.
One can easily verify that the weak formulation corresponding to the equations 3.5.1.1 and 3.5.1.2 under the boundary conditions 3.5.1.3 to 3.5.1.5 is given by:

$$\int_{\Omega} \frac{\partial \delta \mathbf{T}}{\partial Y} \cdot \begin{pmatrix} 1 & 0 \\ 0 & Le \end{pmatrix} \frac{\partial}{\partial Y} \mathbf{T} + U \frac{\partial T}{\partial Gz} \cdot \delta \mathbf{T} d\Omega - \int_{\Gamma_{vapour}} \frac{\partial \delta \mathbf{T}}{\partial Y} \cdot \begin{pmatrix} 1 & 0 \\ 0 & Le \end{pmatrix} \frac{\partial}{\partial Y} \mathbf{T} n_Y d\Gamma = 0 \quad (3.5.1.7)$$

where Γ_{vapour} is the interface between vapour and mixture with the special boundary condition and $\delta \mathbf{T}$ is the test function to be used in the weak formulation. In order to satisfy the boundary condition $\frac{\partial \theta}{\partial Y} = \Lambda \frac{\partial \Gamma}{\partial Y}$ we transform the test function such that the boundary integral vanishes under the boundary condition 3.5.1.4.
If we introduce $\delta \tilde{\mathbf{T}}$ by the transformation

$$\delta \tilde{\mathbf{T}} = \begin{pmatrix} \delta \tilde{\theta} \\ \delta \tilde{\gamma} \end{pmatrix} = \begin{pmatrix} -1 & 0 \\ \frac{\Lambda}{Le} & 1 \end{pmatrix} \tilde{\mathbf{T}} \quad (3.5.1.8)$$

we see that the boundary integral can be written as

$$\int_{\Gamma_{vapour}} \frac{\partial \delta \tilde{\mathbf{T}}}{\partial Y} \cdot \begin{pmatrix} -1 & \frac{\Lambda}{Le} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & Le \end{pmatrix} \frac{\partial}{\partial Y} \mathbf{T} n_Y d\Gamma \quad (3.5.1.9)$$

The first row in 3.5.1.9 vanishes because of boundary condition 3.5.1.4 the second row vanishes because the second unknown is prescribed after transformation. So the test function must be transformed by equation 3.5.1.8. In the input of program SEPCOMP this is done by `matrixv`. Since the transformation of unknowns and test functions is different we need a non-symmetric transformation in this case.

First we give the program that is based upon SEPCOMP. The main program consists only of a call to SEPCOM. The listing for this program is given by:

```
      program absorber
      call sepcom(0)
      end

      function funccf ( ifunc, x, y, z )
      implicit none
      double precision funccf, x, y, z
      integer ifunc
      if ( ifunc==1 ) then

!     --- ifunc = 1, compute u, with respect to teta

         funccf = 1.5*(2*y-y**2)

      else

!     --- ifunc = 2, compute u, with respect to gamma

         funccf = 1.5*(2*y-y**2)

      end if
      end
```

This program needs an input file which is the same as for SEPCOMP. The following input file may be used to solve the problem:

```
#  absorb.prb
#
constants
   reals
      LE     = 2e-3         # LE    =D/a
                            #   Lewis number
      LAMBDA = 1e-2         # LAmBDA =-Le*dH/Cp/A/(1-W0)
                            #   dim.less heat of absorption
      Bi     = 5            # BI    =Ac*delta/lambda
                            #   Biot number cooling/film
      TETAC  = -0.2         # TETAC =(Tc-T0)/(Te-T0)
      TETAC_BI = tetac*bi   # TETAC_BI = TETAC*BI
      Lam_Le = lambda/le    # LAMBDA/LE
   vector_names
      theta
end
*
* problem definition
*
problem
   types
      elgrp1=(type=805)
   natboundcond
      bngrp1=(type=806)

   bounelements
       belm1=curves(shape=1,c1)

   essbouncond
      degfd1,degfd2=curves0(c8)
      degfd2=curves200(c4)

*                    Transformation of the unknowns T and w at the interface

   localtransform
      degfd1,degfd2=curves200(c4),transformation=non_symmetric//
      matrixr=(1,0,-1,1), matrixv=(-1,0, Lam_Le,1)
END
essential boundary conditions
   degfd2, curves200 (c4), value = 1
end
coefficients
   elgr1 ( nparm=45 )
      coef 6 = 0          # diffusion in x-dir (teta equation)
      coef 9 = 1          # diffusion in y-dir
      coef12 = func=1     # velocity  in x-dir
      coef13 = 0          # velocity  in y-dir
      coef21 = 0          # diffusion in x-dir (gamma equation)
      coef24 =  LE        # diffusion in y-dir
      coef27 = func=2     # velocity  in x-dir
      coef28 = 0          # velocity  in y-dir
   bngrp1 ( nparm=25 )
      coef 6 =  BI        # sigma=bi (teta equation)
      coef 7 =  TETAC_BI  # h = tetac bi
end
```

```
structure
   prescribe_boundary_conditions
   solve_linear_system
end
end_of_sepran_input
```

Once the solution has been computed, it may be printed and plotted by the postprocessing program SEPPOST. SEPPOST also requires an input file. The following input file prints the computed solution, makes a standard contour plot as well as a coloured contour plot. In order to identify the plot an extra text identification is submitted.

```
#  absorb.pst
#
post processing

# Define names of vectors

   #  Print both vectors completely

   print  theta

   # PLot the results

   plot contour  theta, degfd=1
   plot contour  theta, degfd=2
   plot coloured contour  theta, degfd=1
   plot coloured contour  theta, degfd=2

end
```

Figure 3.5.1.2 shows the computed isotherms.

3.5.2 An artificial example of the use of periodical boundary conditions to connect two regions

In this section we an artificial example, to show how periodical boundary conditions can be used to connect two regions through boundary conditions. The main difference with the examples in Section 3.1.10 is that we have two unknowns per point and each of these unknowns has different connection boundary conditions.
In order to get this example into your local directory use:

```
sepgetex testperiod04
```

To run this example use

```
sepmesh testperiod04.msh
view mesh by jsepview
seplink testperiod04
testperiod04 < testperiod04.prb
view results by jsepview
```

In this example we consider the following artificial problem.
Let Ω_1 be the unit square ((0,1) $\times$ (0,1)) and Ω_2 be the unit square ((1,1) $\times$ (2,1))
Let each component of the vector $\mathbf{T}$ defined by

$$\mathbf{T} = \begin{pmatrix} T_1 \\ T_2 \end{pmatrix} \tag{3.5.2.1}$$

satisfy the diffusion equation with different diffusion parameters κ in each region, i.e $-$ div $\kappa_1 \nabla T_i = 0$ in Ω_1 and $-$ div $\kappa_2 \nabla T_i = 0$ in Ω_2 $(i = 1, 2)$.
On the lower boundary $(y = 0)$ and the upper boundary $(y = 1)$, as well as the left-hand side of Ω_1 and the right-hand side of Ω_2 we prescribe the temperature components T_i by $T_1(x, y) = y$, $T_2(x, y) = x$ in Ω_1 and $T_1(x, y) = 2y$, $T_2(x, y) = 0.5x + 0.5$ in Ω_2 (Dirichlet boundary condition).
Furthermore we assume that both regions which have separate boundaries for $x = 1$ are coupled through coupling conditions. Since in this case we have two unknowns per point the number of coupling conditions must be twice that used in Section 3.1.10
The coupling boundary conditions we prescribe are for T_1 that the value in the left-hand side of Ω_2 is twice the value in the right-hand side of Ω_1, and that the fluxes on both sides are equal.
For T_2 we assume continuity of the temperature as well as the flux.
So if the curves at $x = 1$ are defined as Cleft and Cright, actually the boundary condition is defined as $T_1\text{Cleft} = 2T_1\text{Cright}$ and $\kappa_1 \frac{\partial T_1}{\partial x}|_{\text{Cleft}} = \kappa_2 \frac{\partial T_1}{\partial x}|_{\text{Cright}}$.
$T_2\text{Cleft} = T_2\text{Cright}$ and $\kappa_1 \frac{\partial T_2}{\partial x}|_{\text{Cleft}} = \kappa_2 \frac{\partial T_2}{\partial x}|_{\text{Cright}}$. This means that we have a periodical boundary conditions for T_2 and a periodical boundary condition with a multiplication factor 2 for T_1.
One easily verifies that if $\kappa_1 = 1$ and $\kappa_2 = 2$, the exact solution is given by $T_1 = y$ in Ω_1, $T_1 = 2y$ in Ω_2, and $T_2 = x$ in Ω_1, $T_x = 0.5 + 0.5x$ in Ω_2

The equation itself is standard, and so are the Dirichlet boundary conditions. The periodical boundary conditions, however, require so-called connection elements, which identify unknowns on Cleft and Cright.
The mesh file used in this case is:

```
#  testperiod04.msh
#
#  mesh file for 2d periodical boundary conditions problem
#  See testperiod04.prb for a description
#
#  To run this file use:
#     sepmesh testperiod04.msh
```

```
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1        # width of the region
      length = 1       # length of the first subregion
      length2 = 2      # length of the second subregion
   integers
      n = 4            # number of elements in length direction
      m = 4            # number of elements in width direction
      shape_cur = 1    # Linear elements along curves
      shape_sur = 5    # Bi-linear quadrilaterals in surfaces
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
     # subregion 1
      p1=(0,0)              # Left under point
      p2=( length,0)        # Right under point
      p3=( length, width)   # Right upper point
      p4=(0, width)         # Left upper point
     # subregion 2
      p11=( length,0)         # Left under point
      p12=( length2,0)        # Right under point
      p13=( length2, width)   # Right upper point
      p14=( length, width)    # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
     # subregion 1
      c1=line  shape_cur (p1,p2,nelm= n)     # lower wall
      c2=line  shape_cur (p2,p3,nelm= m)     # right-hand side
      c3=line  shape_cur (p3,p4,nelm= n)     # upper wall
      c4=line  shape_cur (p4,p1,nelm= m)     # left-hand side
     # subregion 2
      c11=line  shape_cur (p11,p12,nelm= n)     # lower wall
      c12=line  shape_cur (p12,p13,nelm= m)     # right-hand side
      c13=line  shape_cur (p13,p14,nelm= n)     # upper wall
      c14=line  shape_cur (p14,p11,nelm= m)     # left-hand side
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
     # subregion 1
      s1=rectangle  shape_sur (c1,c2,c3,c4)
     # subregion 2
```

```
      s2=rectangle  shape_sur (c11,c12,c13,c14)

#  Coupling of surfaces to element groups

   meshsurf
      selm1 = s1
      selm2 = s2

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2
end
```

Since the boundary conditions depend on the coordinates, we need a main program to define the function.

```
      program testperiod04
      implicit none

!     --- File for  2d periodical boundary conditions problem
!         See testperiod04.prb and the manual Examples Section 3.5.2
!         for a description

      call startsepcomp

      end

!     --- Function funcbc for the essential boundary conditions

      function funcbc ( ichoice, x, y, z )
      implicit none
      integer ichoice
      double precision x, y, z, funcbc
      if ( ichoice==1 ) then
         funcbc = y
      else if ( ichoice==2 ) then
         funcbc = x
      else if ( ichoice==3 ) then
         funcbc = 2d0*y
      else if ( ichoice==4 ) then
         funcbc = 0.5d0*x+0.5d0
      else

!     --- ichoice <1 or > 4: error

         call eropen('funcbc')
         call errint(ichoice,1)
         call errsub ( 1, 1, 0, 0)
         call erclos('funcbc')
         call instop
         funcbc = 0d0

      end if

      end
```

```
!     --- Function func for the creation of the exact solution

      function func ( ichoice, x, y, z )
      implicit none
      integer ichoice
      double precision x, y, z, func, funcbc

      func = funcbc ( ichoice, x, y, z )

      end
```

The input file for the computational part is standard. The only special part is the formed by the elements of type -1 defining the periodical boundary conditions.

```
# testperiod04.prb
#
#  problem file for 2d periodical boundary conditions problem
#  See manual Examples Section 3.5.2
#
#  The problem to be solved consist of two squares of size 1x1:
#  S1: (0,0) x (1,1)
#  S2: (1,0) x (2,1)
#
#  The squares are connected by connection elements
#
#  In S1 the solution of the double diffusion equation is: u = ( y, x )
#  In S2 the solution of the double diffusion equation is: u = (2y, 0.5x+0.5 )
#  Hence in the common interface we have continuity of v and a multiplication
#  factor of 2 for T
#
#  The coefficients for the diffusion equation are different for both squares
#
#  To run this file use:
#     sepcomp testperiod04.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      kappa_1 = 1                 # conductivity in S1
      kappa_2 = 2                 # conductivity in S2
   vector_names
      Temperature
      T_exact
   variables
      error
end
#
#  Define the type of problem to be solved
#
```

```
problem                    # See Users Manual Section 3.2.2

   types                         # Define types of elements,
                                 # See Users Manual Section 3.2.2
      elgrp1=805                 # Type number for double laplacian equation
                                 # See Standard problems Section 3.5
      elgrp2=805                 # Type number for double laplacian equation
                                 # See Standard problems Section 3.5
   essbouncond                   # Define where essential boundary conditions are
                                 # given (not the value)
                                 # See Users Manual Section 3.2.2
      curves(c1)                 # Fixed under wall S1
      curves(c3)                 # Fixed upper wall S1
      curves(c4)                 # left-hand side S1
      curves(c11)                # Fixed under wall S2
      curves(c13)                # Fixed upper wall S2
      curves(c12)                # left-hand side S2
                                 # All not prescribed boundary conditions
                                 # satisfy corresponding stress is zero
   periodical_boundary-conditions
      curves(c2,-c14) degfd1, constant=0, factor=2
        #  T_1, multiplication factor 2
      curves(c2,-c14) degfd2, constant=0, factor=1
        #  T_2, multiplication factor 1

end
#
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#
structure                    # See Users Manual Section 3.2.3
 # Compute the temperature
   prescribe_boundary_conditions,  Temperature &
      degfd1, func=1, curves(c1 to c4)    # curve c2 has no effect
   prescribe_boundary_conditions,  Temperature &
      degfd2, func=2, curves(c1 to c4)    # curve c2 has no effect
   prescribe_boundary_conditions,  Temperature &
      degfd1, func=3, curves(c11 to c14)  # curve c14 has no effect
   prescribe_boundary_conditions,  Temperature &
      degfd2, func=4, curves(c11 to c14)  # curve c14 has no effect
   solve_linear_system,  Temperature
 # Compute and print the error
   create_vector T_exact degfd1, func=1, surface(s1)
   create_vector T_exact degfd2, func=2, surface(s1)
   create_vector T_exact degfd1, func=3, surface(s2)
   create_vector T_exact degfd2, func=4, surface(s2)
   error = norm_dif=3, vector1=Temperature, vector2=T_exact
   plot_colored_levels Temperature, degfd = 1, text = 'T_1'
   plot_colored_levels Temperature, degfd = 2, text = 'T_2'
 # Write the results to a file
   output
   print error
end
#  Define the coefficients for the problems
#  All parameters not mentioned are zero
```

```
#  See Users Manual Section 3.2.6 and Standard problems Section 3.5

coefficients
   elgrp1
                          # Omega_1
      coef6  =  kappa_1    # 6:  Heat conduction equation 1
      coef9  = coef6       # 9:  Heat conduction equation 1
      coef21 =  kappa_1    # 21: Heat conduction equation 2
      coef24 = coef6       # 24: Heat conduction equation 2
   elgrp2
                          # Omega_2
      coef6  =  kappa_2    # 6:  Heat conduction equation 1
      coef9  = coef6       # 9:  Heat conduction equation 1
      coef21 =  kappa_2    # 21: Heat conduction equation 2
      coef24 = coef6       # 24: Heat conduction equation 2
end

end_of_sepran_input
```

3.6 Extended second order real linear elliptic and parabolic equations with two degrees of freedom

In this section we treat some examples corresponding to Section 3.6 of the manual Standard Problems. At this moment the following examples are available:

1D biharmonic equation (3.6.1)This concerns an artificial test problem.

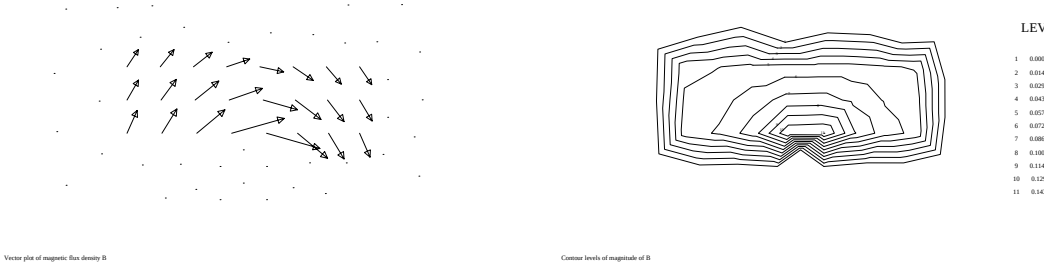


Figure 3.3.2.5: Vector plot of $\mathbf{B}$ and contour plot of $\|\mathbf{B}\|$

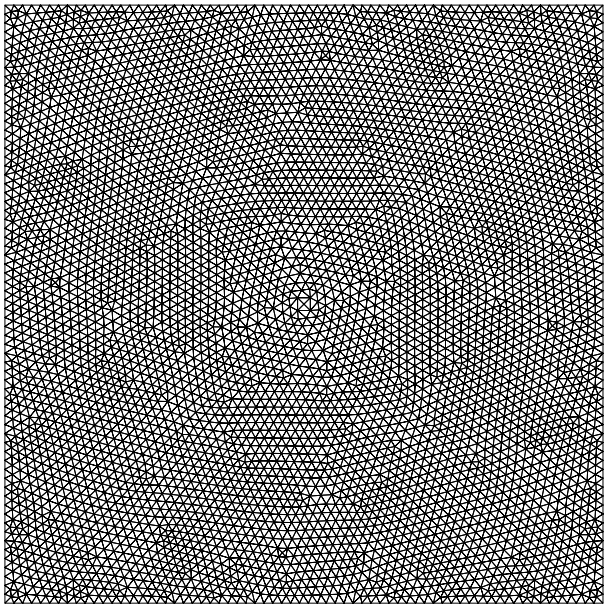
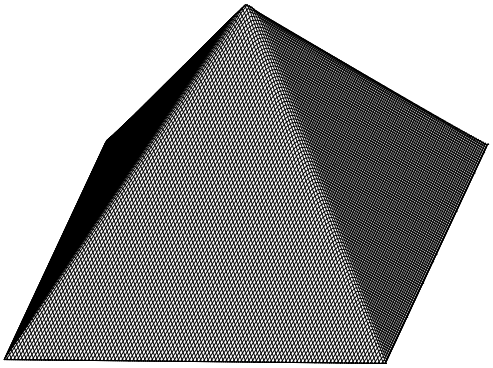
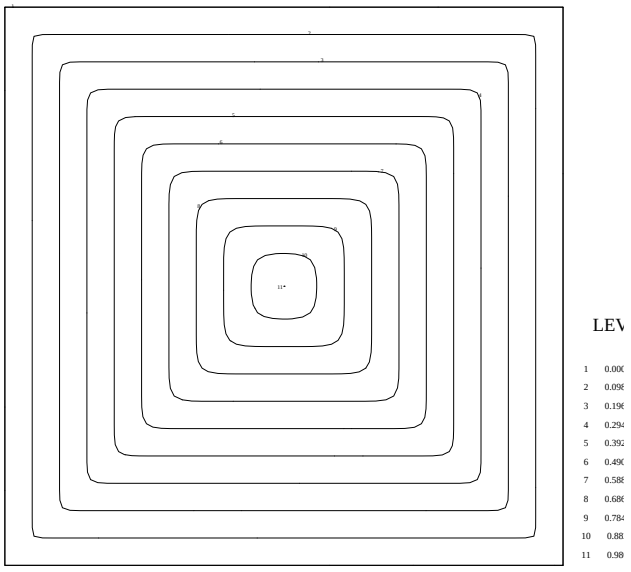


Figure 3.3.3.1: Plot of mesh generated by SEPMESH



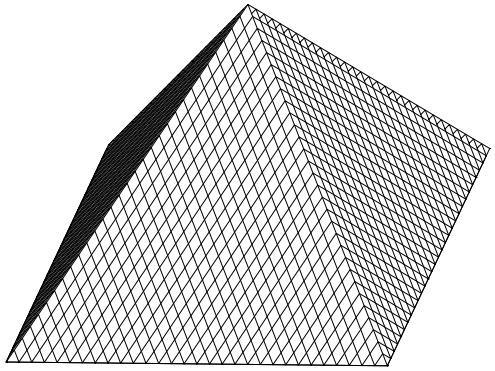
3D plot of oplossing

Figure 3.3.3.2: 3D-plot of solution



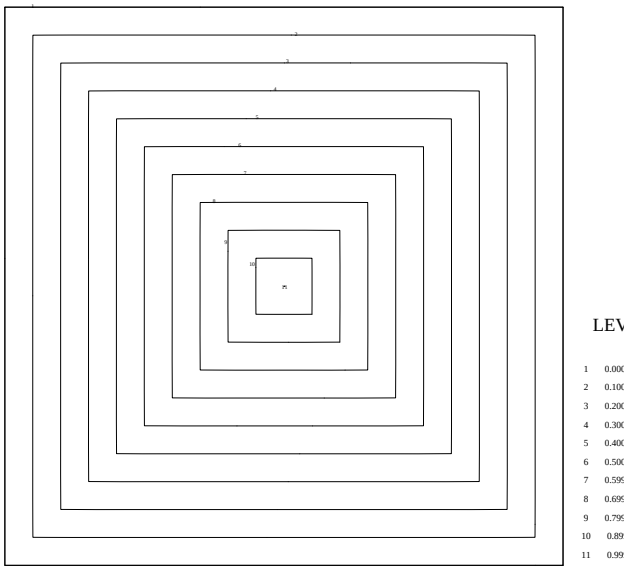
Contour levels of oplossing

Figure 3.3.3.3: Contour plot of solution



3D plot of solution

Figure 3.3.4.1: 3D-plot of solution



Contour levels of solution

Figure 3.3.4.2: Contour plot of solution

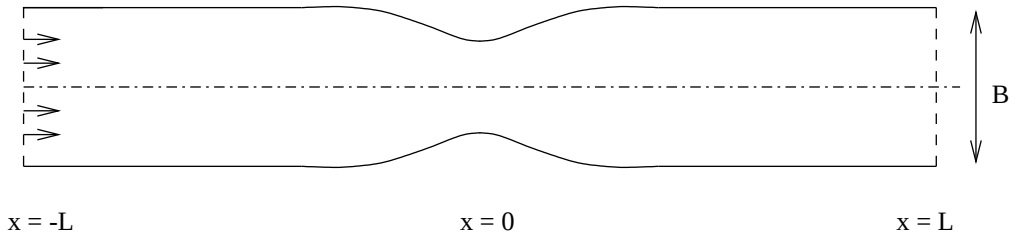


Figure 3.3.5.1: Nozzle

Figure 3.3.5.2: Mesh for nozzle

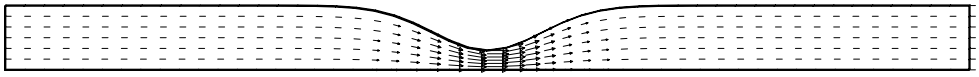


Figure 3.3.5.3: Velocity vectors nozzle

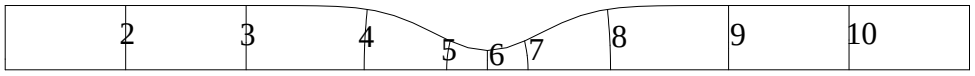


Figure 3.3.5.4: Potential contours in nozzle

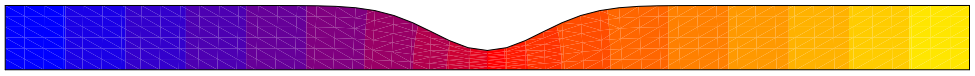


Figure 3.3.5.5: Colored potential levels in nozzle

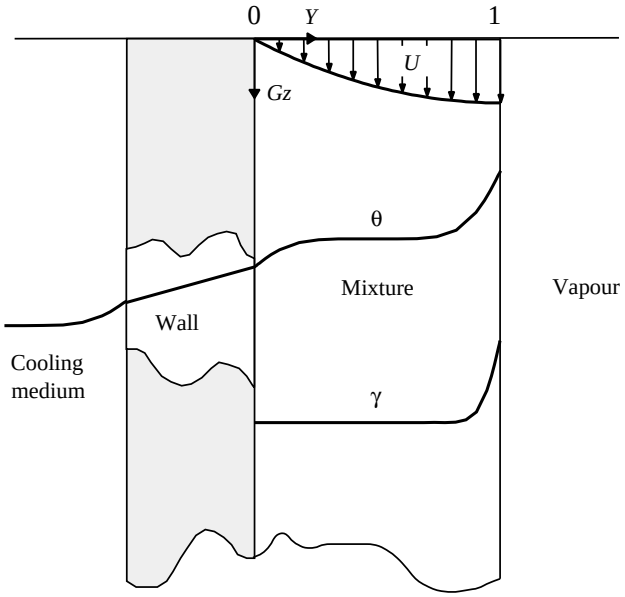


Figure 3.5.1.1: Schematic representation of the cooled vertical wall film absorber

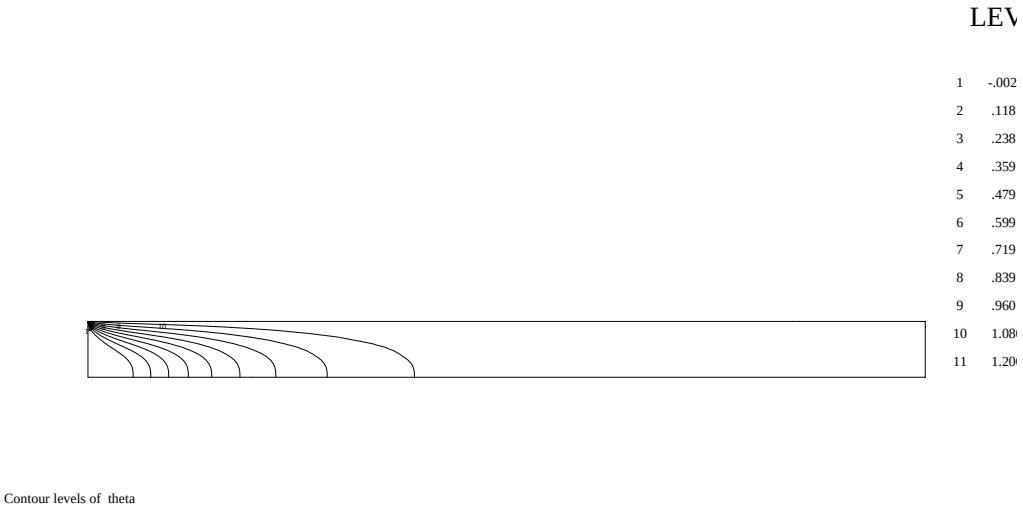


Figure 3.5.1.2: isotherms in the cooled vertical wall film absorber

3.6.1 Example of 1d biharmonic equation, solved as a coupled system of second order equations

As an example of the use of two coupled second order equations, we consider the solution of a 1d biharmonic equation. It concerns an artificial mathematical example.

To get this example into your local directory use:

```
sepgetex testbiharmonisch1d1
```

and to run it use:

```
sepmesh testbiharmonisch1d1.msh
sepcomp testbiharmonisch1d1.prb
```

Consider the 1d biharmonic equation:

$$\frac{\partial^4 u}{\partial x^4} = f \tag{3.6.1.1}$$

with boundary conditions:

$$u \text{and} \frac{\partial^2 u}{\partial x^2} \text{given} \tag{3.6.1.2}$$

Due to the special boundary conditions, this equation can be written as

$$-\frac{\partial^2 u}{\partial x^2} = v \tag{3.6.1.3}$$

$$-\frac{\partial^2 v}{\partial x^2} = f \tag{3.6.1.4}$$

$$\tag{3.6.1.5}$$

or in the form used by the manual Standard Problems:

$$-\frac{\partial^2 u}{\partial x^2} - v = 0 \tag{3.6.1.6}$$

$$-\frac{\partial^2 v}{\partial x^2} = f \tag{3.6.1.7}$$

$$\tag{3.6.1.8}$$

We solve this problem on the region [0,1] with the following boundary conditions:

1. $u(0) = 1, u(1) = 1, v(0) = 0, v(0) = 0$, exact solution: $u = 1, v = 0$
2. $u(0) = 0, u(1) = 1, v(0) = -2, v(0) = -2$, exact solution: $u = x^2, v = -2$
3. $u(0) = 0, u(1) = 1, v(0) = 0, v(0) = -12$, exact solution: $u = x^4, v = -12x^2$

The first two problems are solved exactly and the third one has a small error.

The mesh file for this problem is given by

```
# testbiharmonisch1d1.msh
#
#  mesh file for 1d biharmonic equation
#  See Examples Manual Section 3.6.1
#
#  To run this file use:
#     sepmesh testbiharmonisch1d1.msh
#
```

```
#  Creates the file meshoutput
#
#
#
#  Define some general constants
#
constants
  integers
     n = 10
  reals
     L = 1
end
# Create mesh
mesh1d
   points
      p1 = 0
      p2 =  L
   curves
      c1 = line1 ( p1, p2, nelm =  n )
end
```

and the problem file by

```
# testbiharmonisch1d1.prb
#
#  problem file for 1d biharmonic equation
#  See Examples Manual Section 3.6.1
#
#  To run this file use:
#     sepcomp testbiharmonisch1d1.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      kappa       = 1               # diffusion parameter
   vector_names
      potential
end
#
#  Define the type of problem to be solved
#
problem, sequence_number = 1          # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=808                # Type number for second order elliptic equation
                                # See Standard problems Section 3.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
```

```
                              # See Users Manual Section 3.2.2
      points ( p1, p2)        # Essential boundary conditions on all boundaries
end

# Define the essential boundary conditions
#  See Users Manual Section 3.2.5
#  First problem u = 1

essential boundary conditions, sequence_number = 1
   points p1, degfd1, value = 1
   points p2, degfd1, value = 1
end

#  Second problem u = x^2, v=-2

essential boundary conditions, sequence_number = 2
   points p1, degfd1, value = 0
   points p1, degfd2, value = -2
   points p2, degfd1, value = 1
   points p2, degfd2, value = -2
end

#  Third problem u = x^4, v=-12x^2

essential boundary conditions, sequence_number = 3
   points p1, degfd1, value = 0
   points p1, degfd2, value = 0
   points p2, degfd1, value = 1
   points p2, degfd2, value = -12
end

#  Define the coefficients for Laplacian equation
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients, sequence_number = 1
   elgrp1 ( nparm=65 )     # The coefficients are defined by 65 parameters
      coef6  =  kappa            # a11 = kappa
      coef21 = 1                 # a22 = 1
      coef45 = -1                # beta_21
end

coefficients, sequence_number = 2
   elgrp1 ( nparm=65 )     # The coefficients are defined by 65 parameters
      coef6  =  kappa            # a11 = kappa
      coef21 = 1                 # a22 = 1
      coef31 = 24                # f2 = 24
      coef45 = -1                # beta_21
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#

structure                  # See Users Manual Section 3.2.3
```

```
 # Compute the solution of the first problem (u=1, v=0)
 # First prescribe the essential boundary conditions
   # The sequence number refers to the sequence number used in the
   # essential boundary conditions block
   # Since only one input block is present this information is superfluous
   prescribe_boundary_conditions, potential, sequence_number = 1

 # Next solve the system of equations
   # The sequence number seq_coef refers to the sequence number of the
   # input block coefficients and
   # the sequence number seq_solve refers to the sequence number of the
   # input block solve

   solve_linear_system, potential, seq_coef = 1
   print potential

 # Compute the solution of the second problem (u=x^2, v=-2)

   prescribe_boundary_conditions, potential, sequence_number = 2
   solve_linear_system, potential, seq_coef = 1
   print potential

 # Compute the solution of the third problem (u=x^4, v=-12x^2)

   prescribe_boundary_conditions, potential, sequence_number = 3
   solve_linear_system, potential, seq_coef = 2
   print potential

 # Write the results to a file
   # Since no extra information is used, we have omitted an input block

   output

end
```

3.7 Second order wave equations

At this moment we have only one example:

3.8 An artificial example of the solution of a 2d wave equation

3.8 An artificial example of the solution of a 2d wave equation

Consider the wave equation

$$\frac{\partial^2 u}{\partial t^2} - \Delta u = f \tag{3.8.0.9}$$

with $f = -cos(t)(x + 3y)$.
This equation must be solved on a unit square with initial conditions

$$u(t = 0) = cos(t)(x + 3y) \qquad \frac{\partial u}{\partial t}(t = 0) = 0 \tag{3.8.0.10}$$

and boundary conditions

$$u = cos(t)(x + 3y) \tag{3.8.0.11}$$

on the whole boundary.
One easily verifies that the exact solution of this problem is given by $u = cos(t)(x + 3y)$
To get this example in you local directory use:

```
sepgetex examwave1
```

And to run it use

```
sepmesh examwave1.msh
seplink examwave1
examwave1 < examwave1.prb
seppost examwave1.pst
```

A version in which we use the exact solution and the error of the numerical solution is computed is also available under the name:

```
examwave2
```

This version can be copied and run in exactly the same way as examwave1.
In order to solve this problem we apply the standard finite element discretization with elements of type 800 as described in the manual Standard Problems Section 3.1.
The time discretization we apply is the central difference scheme, that is a special method for second order time derivatives.
The mesh for this problem is standard. The input file is

```
#  examwave1.msh
#
#  mesh file for 2d artificial wave problem
#  See Manual Examples Section 3.7.1
#
#  To run this file use:
#     sepmesh examwave1.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the region
      length = 1             # length of the region
   integers
```

```
      shape_cur = 1          # Type of elements along curves
                             # linear elements
      shape_sur = 3          # Type of elements in surface
                             # Linear triangles
end

#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
   coarse(unit=0.1)    # unit length
#
#  user points, provided with local coarseness
#
   points           # See Users Manual Section 2.2
      p1=(0,0,1)
      p2=( width,0,1)
      p3=( width, length,1)
      p4=(0, length,1)
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1=cline  shape_cur (p1,p2)
      c2=cline  shape_cur (p2,p3)
      c3=cline  shape_cur (p3,p4)
      c4=cline  shape_cur (p4,p1)
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=general  shape_sur (c1,c2,c3,c4)
   plot                          # make a plot of the mesh
                                 # See Users Manual Section 2.2

end
```

In order to define the initial conditions, boundary conditions and right-hand side, we need function subroutines. These are given in the following fortran file:

```
      program examwave1
      call sepcom(0)
      end

!     --- function func for the initial condition

      function func ( ichoice, x, y, z )
      implicit none
      double precision func, x, y, z
      integer ichoice
      double precision t, tout, tstep, tend, t0, rtimdu
      integer iflag, icons, itimdu
      common /ctimen/ t, tout, tstep, tend, t0, rtimdu(5), iflag,
     +               icons, itimdu(8)
```

```
      func =  cos(t) * ( x + 3d0 * y )

      end


!     --- function funccf for the right-hand side:

      function funccf ( ichoice, x, y, z )
      implicit none
      double precision funccf, x, y, z
      integer ichoice
      double precision t, tout, tstep, tend, t0, rtimdu
      integer iflag, icons, itimdu
      common /ctimen/ t, tout, tstep, tend, t0, rtimdu(5), iflag,
     +               icons, itimdu(8)

      funccf = -cos(t) * ( x + 3d0 * y )

      end


!     --- function funcbc for essential boundary conditions


      function funcbc ( ichoice, x, y, z )
      implicit none
      double precision funcbc, x, y, z
      integer ichoice
      double precision t, tout, tstep, tend, t0, rtimdu
      integer iflag, icons, itimdu
      common /ctimen/ t, tout, tstep, tend, t0, rtimdu(5), iflag,
     +                icons, itimdu(8)

      funcbc = cos(t) * ( x + 3d0 * y )

      end
```

In the input file for this program we need to define two vectors: the function u and its time derivative, which is stored in un. The input file is given by

```
# examwave1.prb
#
#  problem file for 2d artificial wave problem
#  See Manual Examples Section 3.7.1
#
#  To run this file use:
#     sepcomp examwave1.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
```

```
constants           # See Users Manual Section 1.4
   reals

      t0    = 0           # initial time
      t1    = 1           # end time
      dt    = 0.05        # time step
      tout0 =  t0         # first time for output to sepcomp.out
      tout1 =  t1         # last time for output to sepcomp.out
      dtout =  2*dt       # time step for output to sepcomp.out

   vector_names
      u             ! Contains solution
      un            ! Contains solution at prior time level
                    ! At the start it contains the time-derivative at t = t0
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2

      elgrp1 = 800             # General second order equation
                               # See Standard problems Section 3.1
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      curves(c1 to c4)         # all outer boundaries are prescribed
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   storage_method = compact, symmetric
                  # Symmetric matrix, stored as compact matrix
end

#  Create initial conditions
#  See Users Manual Section 3.2.5
create vector 1
   func = 1                  ! u at t=0 (function)
create vector 2
   value = 0                 ! du/dt at t=0 (derivative)
end
#
#  Essential boundary conditions
#
essential boundary conditions
   curves(c1 to c4),(func=1)      # Boundary contions are only necessary for u
                                  # They depend on time and place, hence a
                                  # function is used
end
```

```
#  Define the coefficients for the wave equation
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients
   elgrp1(nparm=20)
      coef6  = 1                  # a11 = 1  (laplace)
      coef9  = coef 6             # a22 = 1
      coef16 = func = 1           # f is a function
      coef17 = 1                  # rho = 1
end

#  Define input for the time integration

time_integration
   method = central_differences        ! standard method for second order
                                       ! time derivatives
   tinit =  t0                         ! initial time
   tend  =  t1                         ! end time
   tstep =  dt                         ! time step
   toutinit =  tout0                   ! initial time for output
   toutend  =  tout1                   ! end time for output
   toutstep =  dtout                   ! time step for output
   seq_coefficients = 1                ! defines which coefficients must
                                       ! be used
   diagonal_mass_matrix                ! The mass matrix is diagonal
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved

structure
  # Fill initial condition in u and derivative at t=0 in un
   create_vector u
  # Integrate the equation for t0 to tend
   solve_time_dependent_problem, u
end

end_of_sepran_input
```

Finally we can plot the solution using the following input file for seppost.

```
# examwave1.pst
#  Input file for postprocessing for 2d artificial wave problem
#  See Manual Examples Section 3.7.1
#
#
#  To run this file use:
#     seppost examwave1.pst > examwave1.out
#
#  Reads the files meshoutput and sepcomp.out
#
#
postprocessing                    # See Users Manual Section 5.2
   time = (0,1)
```

```
      print u
      plot contour u
   time history plot point(.5,.5) u
end
```

4 Elements for lubrication theory

4.1 The Reynolds equation

4.1.1 Oil lubricated radial sliding bearing (Reynolds equation)

Consider an oil lubricated radial bearing with eccentricity e. In Figure 4.1.1.1 the cross-section of the bearing has been sketched. The oil film thickness is small and therefore the Navier-Stokes equations describing the flow may be approximated by the Reynolds equation for the pressure. See 4.1.
In order to get this example into your local directory use:

```
sepgetex bearing
```

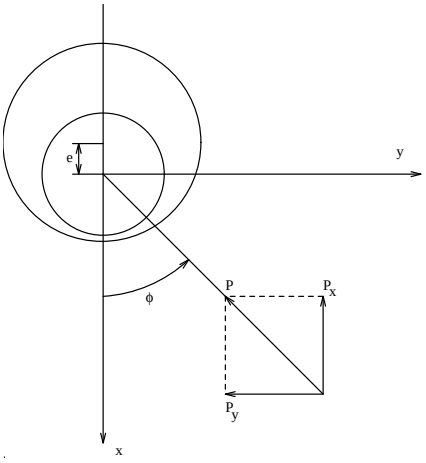


Figure 4.1.1.1: Cross-section of the oil lubricated radial sliding bearing

The computational region is mapped onto a rectangle, where the computational x-co-ordinate is equal to the parameter along the surface of the bearing in ϕ-direction ($0 \leq x \leq \pi D$, D diameter of the bearing). The computational y-co-ordinate is equal to the z-co-ordinate of the bearing. Since the solution is symmetric with respect to the plane z = L/2, only one half of the bearing is computed.
The thickness of the film is described by the function h.
The following parameters are used in the computation:

$$
\begin{aligned}
h(\phi) &= 8.2 \times 10^{-5}(1 - e\cos(\phi))\ m \\
D &= 0.05\ m \\
\tfrac{2e}{D} &= 0.4 \\
\mu &= 0.04\ Ns/m^2 \\
h_t &= 0\ m/s \\
k &= 0\ m^3/Ns \\
p_0 &= 0\ N/m^2 \\
u_1 &= 2.04\ m/s \qquad\qquad u_2 = 0\ m/s
\end{aligned}
$$

The mesh is generated by the mesh generator SEPMESH and consists of a rectangle with sides C1 to C4. The midpoints of the lower and upper sides are used to define a larger coarseness than for the end points. For the mesh we refer to the file `bearing.msh`.
The following boundary conditions are used:

Lower boundary C_1: no flow $\quad(\frac{h^3}{12\mu}\frac{\partial p}{\partial n} - \frac{h}{2}\mathbf{u}\cdot\mathbf{n} = 0)$

Upper boundary C_3 of the bearing: boundary pressure $p = 0$

Curves $C_2 4$ respectively C_4: Since these boundaries coincide with $\phi = 0$ respectively $\phi = 2\pi$ we need periodical boundary conditions to couple both boundaries.

If the Reynolds equation is solved by the preceding parameters, we get negative pressures (cavitation). In order to prevent the negative pressures we the need the extra constraint:

$$p \geq 0$$

Important physical parameters are the load $\mathbf{f} = (f_x, f_y)^T$ and the attitude angle $\phi = \arctan(\frac{f_y}{f_x})$, with $f_x = \int\limits_\Omega -p\cos(\phi)d\Omega$, and $f_y = \int\limits_\Omega -p\sin(\phi)d\Omega$. Both parameters are computed in the subroutine LOAD.

Solution procedure

In order to satisfy the constraint $p \geq 0$,we have the option to apply either the linear solver using constrained overrelaxation or Kumars mass conservation method. In this section we solve the problem in three different ways, all giving the same results:

1. Solving the standard Reynolds equation described in the manual "Standard Problems" Section 4.1, using constrained overrelaxation.
2. Solving the Reynolds equation as a special case of the second order elliptic equation described in the manual "Standard Problems" Section 3.1, using constrained overrelaxation.
3. Solving the standard Reynolds equation described in the manual "Standard Problems" Section 4.1, using Kumars mass conservation method.

Standard Reynolds equation using constrained overrelaxation

As starting value the solution of the Reynolds equation without constraint is used.

The region is subdivided into triangles by the submesh generator "RECTANGLE". As an example linear triangles have been used.
The input file for sepmesh can be found in the directory `$SPHOME/sourceexam/bearing` The input file for sepcomp is given by:

```
#  bearing.prb
#
#  problem file for Oil lubricated radial sliding bearing
#
#  To run this file use:
#     sepcomp  bearing.prb
#
#  Uses the file meshoutput
#
#  Define some general constants
#
constants

   reals
      mu = 0.04        ! viscosity
      u1 = 2.04        ! velocity in x-direction
      diam   = 0.05    ! diameter
      deltar = 8.2e-5  ! delta_r
      eps    = 0.4     ! eccentricity

end

*  Problem definition

problem
   reynolds                         # Reynolds equation
   periodical_boundary_conditions
      curves (c2,c4)
   essential_boundary_conditions    # Positions where essential boundary
      curves(c3)                    # conditions are given
end

*  Structure of the program

structure

#  First part: without the effect of cavitation

   matrix_structure compact, symmetric  ! an iterative method is used
   vector pressure = 0          ! create and clear pressure vector

   phi = x_coor*2/diam          ! phi is a vector depending on x
   layer_thickness = deltar*(1-eps*cos(phi))  ! h is a vector depending on phi
   viscosity = mu
   u_velocity = u1
```

```
   solve_linear_system  pressure  ! The standard preconditioned CG method
                                  ! is applied

# Compute load and attitude angle

   fx = integral ( -pressure*cos(phi) ) ! / -p cos(phi) d Omega
   fy = integral ( -pressure*sin(phi) ) ! / -p sin(phi) d Omega
   ftot = sqrt(fx**2+fy**2)
   angle = atan(fy/fx)
   print_text 'No effect of cavitation'
   print fx  , text = ' horizontal component of load'
   print fy  , text = ' vertical component of load  '
   print ftot, text = ' modulus of load     '
   print angle , text = ' attitude angle      '

#  Prints and plots

   plot_contour pressure
   plot_coloured_levels pressure

#  Second part: with the effect of cavitation

   press_pos = pressure

   matrix_structure row_compact    ! use overrelaxation

   sol_minimum = 0                 ! constraint: p>=0
   solve_linear_system, press_pos

# Compute load and attitude angle

   fx = integral ( -press_pos*cos(phi) ) ! / -p cos(phi) d Omega
   fy = integral ( -press_pos*sin(phi) ) ! / -p sin(phi) d Omega
   ftot = sqrt(fx**2+fy**2)
   angle = atan(fy/fx)
   print 'With effect of cavitation'
   print fx  , text = ' horizontal component of load'
   print fy  , text = ' vertical component of load  '
   print ftot, text = ' modulus of load            '
   print angle , text = ' attitude angle             '

#  Prints and plots

   plot_contour press_pos
   plot_coloured_levels press_pos
   no_output

end
```

This program needs an input file which is the same as for SEPCOMP. Since the solution procedure is more complex than the standard solution of linear problems, the structure of the program must also be defined in the input file.

The structure of the program consists of the following steps:

- The linear problem is solved without constraints. The system of equations is solved by a preconditioned CG algorithm. As a consequence the structure of the matrix is defined by method = 5.
  The result of the computation is stored in `pressure`.
- The load and the attitude angle are computed and printed
- The linear problem is solved with constraints. At this moment only overrelaxation with constraints is available. Since this method requires a structure defined by method = 9, the structure of the matrix must be recomputed. The result of the computation is stored in `press_pos`, the result stored in `pressure` is used as starting vector and hence must be copied in the second vector first.
- The new load and the attitude angle are computed and printed
- Both vectors computed are plotted.

Figure 4.1.1.2 shows the contour plots for the first approximation, Figure 4.1.1.3 for the final solution. Both plots may be visualized by the program SEPVIEW.

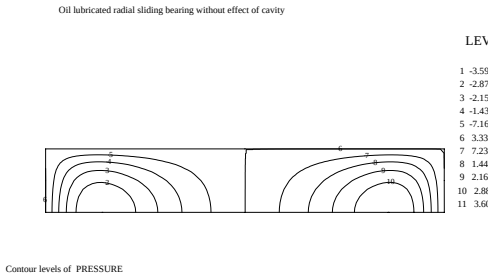


Figure 4.1.1.2: Isobars generated by SEPCOMP with cavity not taking into account

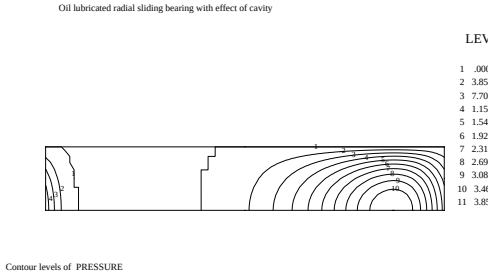


Figure 4.1.1.3: Isobars generated by SEPCOMP with cavity taking into account

4.1.2 Oil lubricated radial sliding bearing solved by general elliptic equation

Instead of using the standard Reynolds element, the same results can be achieved by using the standard element for second order elliptic equations.
The mesh file is identical to the one in Section 4.1 but the problem file is a little bit different.
In order to get this example into your local directory use:

```
sepgetex bearing_elliptic
```

To solve the Reynolds equation we consider the general elliptic equation from the manual Standard Problems Section (3.1).
The translation into the elliptic equation is as follows:
The diffusion is equal to $h^3/(12\mu)$.
The γ vector is equal to (-uh/2,0).
All other terms in the general elliptic equation are zero. We shall not print the input file `bearing_elliptic.prb` below but consider only the differences with the file `bearing.prb`.
First of all the type number `reynolds` is replaced by `general_elliptic_equation`.
Further more the filling of coefficients changes to

```
phi = x_coor*2/diam          ! phi is a vector depending on x
h = deltar*(1-eps*cos(phi))  ! h is a vector depending on phi
diffusion = h**3/(12*mu)
x_gamma = -u1*h/2
```

The rest of the file is identical to the file `bearing.prb`.

4.1.3 Oil lubricated radial sliding bearing using Kumars algorithm

To get the corresponding files into your directory use

```
sepgetex bearingmasscons
```

The mesh file is exactly the same as in Section 4.1.
The the problem file differs a little bit as can be seen below

```
#
#      File:  bearingmasscons.prb
#
#      Contents:  Input for program bearingmasscons described in section 4-1-3
#                 the manual examples
#                 Oil lubricated radial sliding bearing
#                 Kumars mass conservation scheme is used
#
#
#
#  To run this file use:
#     sepcomp bearingmasscons.prb
#
#  Reads the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      maxiter = 10           # maximum number of iterations
   reals
      mu = 0.04              # viscosity
      u1 = 2.04              # velocity
      diam   = 0.05          # diameter of bearing
      deltar = 8.2d-5        # maximum height of film
      eps    = 0.4           # eccentricity
      p_cavity = 0           # cavitation pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   reynolds                          # Reynolds equation
   periodical_boundary_conditions
      curves (c2,c4)
   essential_boundary_conditions     # Positions where essential boundary
      curves(c3)                     # conditions are given
      cavitation 1             # Pressure is prescribed in cavitation region

end
#
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#
structure                  # See Users Manual Section 3.2.3
```

```
   matrix_structure compact, symmetric  ! an iterative method is used

 # Compute the potential
 # First prescribe the essential boundary conditions

   vector pressure = 0          ! create and clear pressure vector

   phi = x_coor*2/diam          ! phi is a vector depending on x
   layer_thickness = deltar*(1-eps*cos(phi))  ! h is a vector depending on phi
   viscosity = mu
   u_velocity = u1

 # Next compute pressure by Kumars algorithm

   solve_bearing

# Compute load and attitude angle

   fx = integral ( -pressure*cos(phi) ) ! / -p cos(phi) d Omega
   fy = integral ( -pressure*sin(phi) ) ! / -p sin(phi) d Omega
   ftot = sqrt(fx**2+fy**2)
   angle = atan(fy/fx)
   print fx  , text = ' horizontal component of load'
   print fy  , text = ' vertical component of load  '
   print ftot, text = ' modulus of load            '
   print angle , text = ' attitude angle             '

#  Prints and plots

   plot_contour pressure
   plot_coloured_levels pressure
   no_output

end
```

4.1.4 Compressible slider bearing

In this example we consider the one-dimensional slider bearing. This example is a good choice to verify the validity of the finite element method, since the exact solution for this problem can be found in Harrison (1913).
In Figure 4.1.4.1 the cross-section of the bearing has been sketched. The film thickness is small and therefore the Navier-Stokes equations describing the flow may be approximated by the Reynolds equation for the pressure. See 4.1. As lubricant air is used, which means that the compressible version of the Reynolds equation must be solved.

In order to get this example into your local directory use:

```
sepgetex bearing1
```

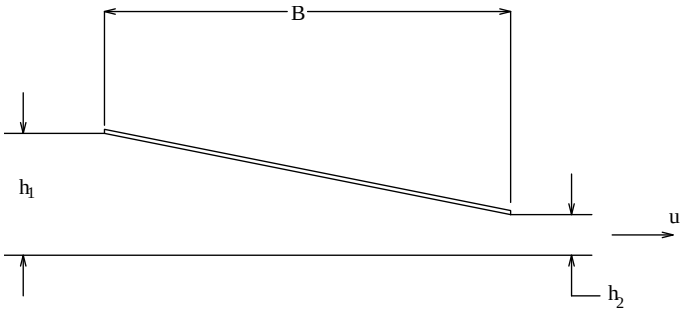


Figure 4.1.4.1: Cross-section of the air lubricated one-dimensional sliding bearing

Since it is known that the pressure has a steep gradient near the minimum height of the film, the mesh is refined in the last part of the region. Figure 4.1.4.2 shows the mesh used in the computation. This mesh has been created by program SEPMESH using the following input.

```
#  bearing1.msh
#
#  mesh file for Air lubricated radial sliding bearing
#
#  To run this file use:
#     sepmesh bearing1.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants
   integers
      n = 20             ! number of elements for first and last part
   reals
      mid = 0.7          ! split of first and second part
      length = 1         ! length of bearing
end
#
#  Define the mesh
#
```

```
mesh1d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2

      p1=0
      p3=mid
      p2=length
   curves
      c1=line (p1,p3,nelm=n)
      c2=line (p3,p2,nelm=n)
   plot, nodes = 1
end
```

Figure 4.1.4.2: Mesh for one-dimensional slider bearing

The test example described in Harrison is dimensionless, but in SEPRAN we have to define values for the physical parameters. In order to get exactly the same dimensionless parameter as Harrison, the following choices have been made:

$B = 1$
$h = 0.3 - 0.2x$, i.e. $h_1 = 0.3$ and $h_2 = 0.1$
$\mu = 0.0166666$
$h_t = 0$
$k = 0$
$p_0 = 0$
$u_1 = 1$

The atmospheric pressure p_a is normalized to 1, hence the computed pressure indicates the ratio $\frac{p}{pa}$. Mark that for the compressible bearing the pressure must always be positive, since otherwise the non-linear algorithm fails.

The compressible Reynolds equations are non-linear, so a non-linear solver must be used. Such a solver always requires a starting value. In this particular example the solution of the incompressible bearing defines a nice starting value. If we compute the solution of this bearing first, we have also the opportunity to compare the pressures computed by the incompressible and the compressible Reynolds equations.

Since the solution procedure is more complex than the standard solution of linear or non-linear problems, the structure of the program must also be defined in the input file.

The structure of the program consists of the following steps:

- The linear problem is solved. The solution is stored in vector 1. A direct method is used for this one-dimensional problem.
- The solution is copied from vector 1 to vector 2. In this way both vector 1 and vector 2 can be plotted.

- The non-linear problem is solved, where the copied vector is used as starting value. The equations are linearized by a newton linearization. This example took only 2 iterations to converge to the final solution.
- Both vectors computed are written to the output file sepcomp.out for post-processing purposes.

The following input file may be used to solve the problem:

```
#  bearing1.prb
#
#  problem file for Air lubricated radial sliding bearing
#
#  To run this file use:
#     sepcomp bearing1.prb
#
#  Uses the file meshoutput
#
#  Define some general constants
#
constants

   reals
      viscosity = 0.0166666
      velocity = 1
end

*  Problem definition

problem
   types
      Reynolds
   essbouncond
      points(p1 to p2)
end


*  Structure of the program

structure

#   Define layer_thickness h

   h = 0.3-0.2*x_coor

   layer_thickness = h

#   First solve incompressible (linear) system
```

```
   prescribe_boundary_conditions press_incp = 1
   solve_linear_system press_incp

#   Next compressible (non-linear) system
#   Use incompressible pressure as start

   press_comp = press_incp
   type_of_bearing = 'compressible'

   solve_nonlinear_system press_comp, print_level=1

   print press_incp
   print press_comp
   plot_function press_incp, press_comp
   no_output

end
```

Figure 4.1.4.3 shows the pressure plot made by program SEPCOMP.

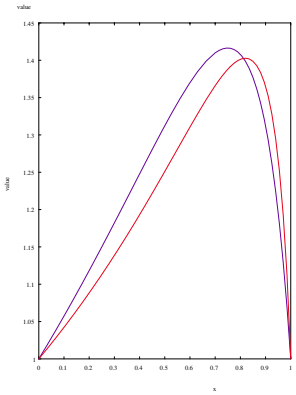


Figure 4.1.4.3: Pressures generated by SEPCOMP. blue incompressible, red compressible

4.1.5 A hydrostatic thrust bearing

Consider an externally pressurized, water lubricated, circular thrust bearing sliding on a track (see Figure 4.1.5.1). The pressure in the thin water film between the bearing and the track can be

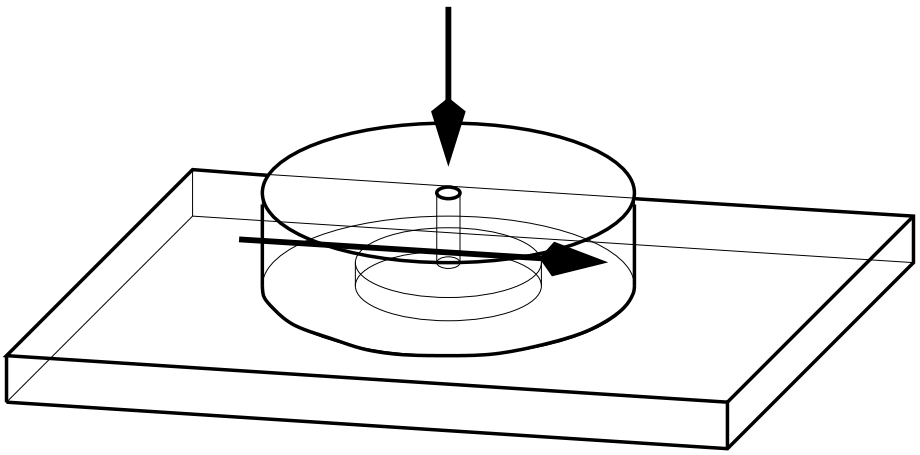


Figure 4.1.5.1: Thrust bearing

described using the Reynolds equation. The water is fed by a pump through a resistor into the central recess of the bearing (see Figure 4.1.5.2). We want to calculate the load capacity and the

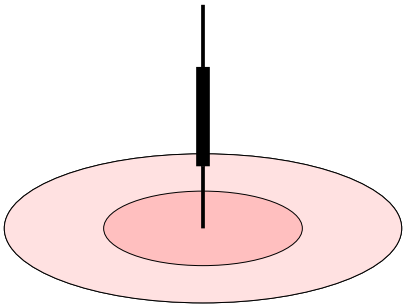


Figure 4.1.5.2: Film geometry

flow of this bearing given certain operating conditions. The parameters used in the computation are given in table 4.1.5.1.

Bearing diameter	D	0.740	m
Water film height	h	$0.1\ 10^{-3}$	m
Recess diameter	D_R	0.530	m
Recess height	h_R	$5.0\ 10^{-3}$	m
Water viscosity	μ	0.001	Ns/m^2
Bearing velocity	u_1	0.25	m/s
Supply resistor	γ	$0.6\ 10^{-6}$	$m^4/N^{1/2}s$
Supply pressure	p_S	$15.0\ 10^5$	N/m^2

Table 4.1.5.1: Parameters

To get this example into your local directory give the command:

```
sepgetex hydrostat_thrust
```

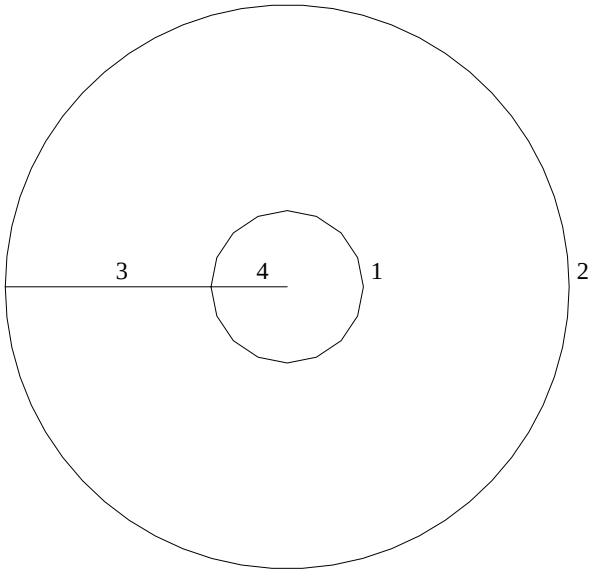


Figure 4.1.5.3: Definition of curves

The mesh is generated by the mesh generator SEPMESH. Figure 4.1.5.3 shows the curves in the mesh. The input file for the mesh generator is given below:

```
#   hydrostat_thrust.msh
#
#   Circular Thrust Bearing, 1 recess
#
#   Contents: Mesh file for hydrostatic thrust bearing
#   See Manual Standard Elements Section 4.1.5
#
#   Author: R.A.J. van Ostayen
#     Date: 21-11-97
#
#
#  To run this file use:
#     sepmesh hydrostat_thrust.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
  reals
    Rb = 0.370      # Radius of the thrust bearing
    Rr = 0.100      # Radius of the bearing recess

    Cc = 1.0        # Coarse value (centre)
    Cb = 1.0        # Coarse value (bearing)
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
  coarse (unit = 0.04)  # The unit length = 0.04
                        # Coarseness is applied
#
#  user points
#
  points           # See Users Manual Section 2.2

    p1  = ( 0.0, 0.0, Cc)       # Centre of the bearing

    p2  = ( -Rb, 0.0, Cb )     # At most left point of bearing
    p3  = ( -Rr, 0.0, Cb )     # At most left point of bearing recess

#
#  curves
#
   curves           # See Users Manual Section 2.3

    c1  = carc  ( p3, p3, p1 )   # Boundary of bearing recess
    c2  = carc  ( p2, p2, p1 )   # Boundary of bearing
    c3  = cline ( p2, p3)        # Help line to connect bearing and bearing
                                 # recess ( not necessary if triangle is used)
    c4  = cline ( p3, p1)        # Connection line between bearing recess
                                 # and centre point. This is necessary since
                                 # the centre point must be a nodal point
#
```

```
#  surfaces
#
   surfaces         # See Users Manual Section 2.4

    s1  = general 3 ( c1, c4, -c4 )        # Bearing recess
    s2  = general 3 ( c2, c3, -c1, -c3 )   # Bearing minus bearing recess

#  Connect surfaces to element groups

  meshsurf
    selm1 = s1      # Bearing recess
    selm2 = s2      # Bearing minus bearing recess

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2

end
```

Figure 4.1.5.3 shows the mesh generated by SEPMESH. The following boundary conditions are

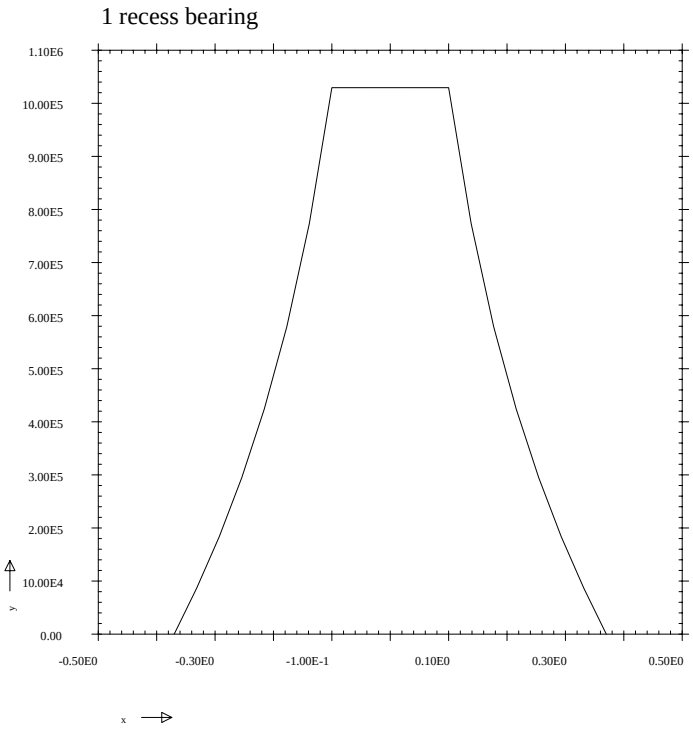


Figure 4.1.5.4: Thrust bearing

used:

- Outer edge: Curves C_1, C_2, C_3 and C_4:
  Essential boundary condition: pressure $p = 0$.

- Center point: Point P_1:
  Natural boundary condition: inertial (= non-linear) supply resistor $q = \gamma\sqrt{p_S - p_R}$

The calculation is performed by the program SEPCOMP. Due to the non-linear supply resistor an iterative procedure is used. The flow through the bearing can be calculated using 4 methods:

- Summing the reaction forces on the outer edge of the bearing (Curves C_1 to C_4).
- Summing the surface flow (icheld=22) on the outer edge.
- Integrating the flow vector (icheld=23) normal to the outer edge.
- Calculating the flow through the supply resistor using the calculated pressure fall across the resistor.

In the input file for the calculation a number of bearing properties are calculated: the load, the flow and the friction forces on the bottom and top surfaces. The input file is given below:

```
#   hydrostat_thrust.prb
#
#   Circular Thrust Bearing, 1 recess
#
#   Contents: Problem file for hydrostatic thrust bearing
#   See Manual Standard Elements Section 4.1.3
#
#   Problem is stationary and non-linear
#
#   Author: R.A.J. van Ostayen
#     Date: 21-11-97
#
#  To run this file use:
#     sepcomp hydrostat_thrust.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4

   reals
      Hf = 0.1e-3     # film height [m]
      Hr = 5e-3       # recess height [m]
      vH2O = 0.001    # viscosity water [Ns/m2]
      U = 0.25        # velocity [m/s]
      Ps = 15e5       # supply pressure [N/m2]
      G = 0.6e-6      # resistor value (non-linear)
end

#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2
```

```
   reynolds
   bounelements                # Defines where the natural boundary conditions
                               # are present
      belm 1 = points (p1)     # The resistor is only present in the center
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
      curves (c2)              # Outer boundary of bearing
end
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because some special integrals and derivatives are computed
#

structure

  matrix_structure, storage_scheme = compact, symmetric, reaction_force

 # initialize solution vector

  create_vector pressure, surfaces (s1), value = 10e5
       # The pressure in the recess is set to 10^6

 # Coefficients for the Reynolds equation

  u_velocity = U
  viscosity = vH2O            # viscosity of water
  restriction = 'non_linear'  # type of restriction relation
  capillary_restriction =  G  # capillary restriction coefficient gamma
  oil_supply_pressure =  Ps   # Water supply pressure

 # Compute the pressure by solving a non-linear system
 # Non-linearity due to resistor

  solve_nonlinear_system pressure, reaction_force = flow_through_surface &
     maxiter = 50, accuracy = 1d-4, criterion = relative, print_level = 2

 # Compute flow in film (vector)

  inplane_force = derivatives ( pressure, icheld = 23 )

 # Compute traction on bottom surface

  bottom_surface_traction = derivatives ( pressure, icheld = 24 )

 # Compute traction on top surface

  top_surface_traction = derivatives ( pressure, icheld = 25 )

 # calculate load, flow and friction

  load = integral ( pressure )
  flow = boundary_sum (flow_through_surface, curves (c2) )

  x_friction_bottom = integral ( bottom_surface_traction, degfd 1 )
  y_friction_bottom = integral ( bottom_surface_traction, degfd 2 )
```

```
  x_friction_top = integral ( top_surface_traction, degfd 1 )
  y_friction_top = integral ( top_surface_traction, degfd 2 )

 # Print the load, flow and friction

  print load,              text = '                      Load [N]: '
  print flow,              text = '     Flow               [m3/s]: '
  print x_friction_bottom, text = 'Friction force (x, bottom) [N]: '
  print y_friction_bottom, text = 'Friction force (y, bottom) [N]: '
  print x_friction_top,    text = '   Friction force (x, top) [N]: '
  print y_friction_top,    text = '   Friction force (y, top) [N]: '

  plot_contour pressure
  plot_3D pressure
  plot_contour flow_through_surface
  plot_contour inplane_force
  plot_vector bottom_surface_traction
  plot_vector top_surface_traction

  no_output

end

# coefficients for Reynolds equation
#  Only those that depend on the element groups

coefficients
  elgrp 1                     # coefficients for the bearing recess
    layer_thickness = Hr      # height of the recess
  elgrp 2                     # coefficients for the rest of the bearing
    layer_thickness = Hf      # film height
end

end_of_sepran_input
```

A contour plot of the calculated pressure is shown in Figure 4.1.5.5. The pressure along a centerline of the bearing is shown in Figure 4.1.5.6.

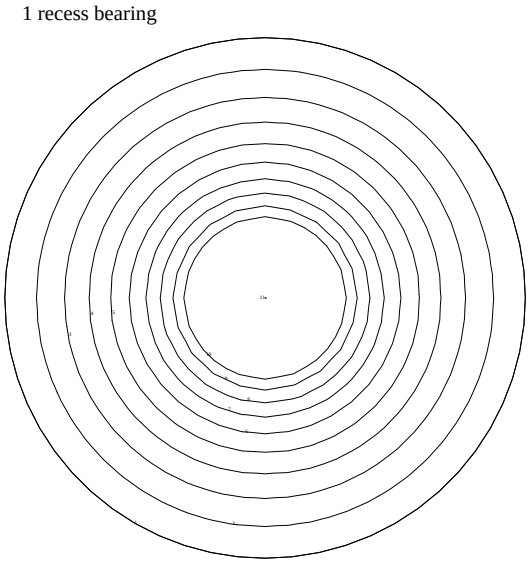


Figure 4.1.5.5: Pressure iso-lines

4.2 Coupled elasticity-flow interaction for a bearing (Reynolds equation coupled with mechanical elements)

4.2.1 Example: the elasto-hydrodynamic lubrication of an oil pumping ring seal

Consider the pumping ring and scraper as given in Figure 4.2.1.1. The pump forms a part of the Philips Stirling engine (See van Heyningen and Kassels 1987). The geometry of the pumping ring is given in Figure 4.2.1.2 For the computations the axi-symmetric model given in Figure 4.2.1.1 has been used.

For the generation of the mesh and the boundary conditions we define 6 user points, 6 curves and one surface. See Figure 4.2.1.2 for an definition.

The problem to be computed is time-dependent. In S1 the axi-symmetric elasticity equations must be satisfied. See 5.1. A linear triangular element with type number 250 is used. At curve C6 the elasticity-Reynolds element is used, since the pressure satisfies the Reynolds equations (time-dependent). Furthermore the following boundary conditions must be satisfied:

C1: $T_r = 0, T_z = 6 \times 10^6\ N/m^2$
C2: $T_r = -11 \times 10^6\ N/m^2, T_z = 0$
C3, C5: $T_r = 0, T_z = 0$
C4: displacement $\mathbf{u} = \mathbf{0}$
P6: $p = 0$
P1: $p = 6 \times 10^6\ N/m^2$

At $t = 0$ a velocity $U = 0$ is assumed. The initial condition is found by solving the non-linear stationary equations by the Newton iteration. As starting value for the iteration we use a zero displacement $\mathbf{u} = \mathbf{0}$ and a linear varying pressure.

The following parameters are used in the computation:

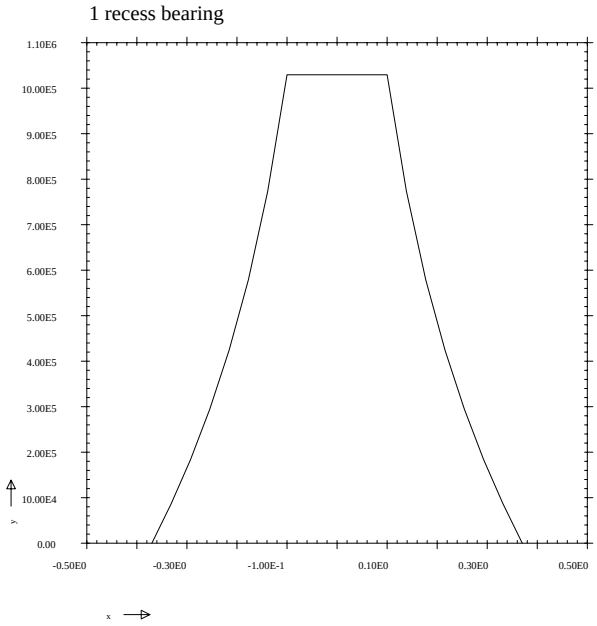


Figure 4.1.5.6: Pressure along a centerline

Seal length	$L = 8\ mm$
Rod diameter	$D = 12\ mm$
Seal thickness	$d = 1\ mm$
Clearance	$h_0 = 8\ \mu m$
Oil viscosity	$\eta = 0.0278\ Ns/m^2$
Young's modulus	$E = 5.27 \times 10^{10}\ N/m^2$
Poisson's ratio	$\nu = 0.44$

In the program all quantities are given in μN and μm, because of the small film thickness.

For t ¿ 0, the problem becomes time-dependent with a velocity U given by:

$$U = \hat{u}sin(\omega t) \text{ m/s}$$

with $\omega = 151.8$ rad/s and $\hat{u} = 3.492$ m/s.

During the time-dependent part, the pressure may become negative (cavitation). Since negative pressures are not physical a so-called "Reynolds" boundary condition is realized by the non-negativity constraint:

$$p \leq 0$$

This constraint is imposed in the program by using subroutine OVERCS (overrelaxation with constraint; see Programmers Guide 6.10.1). To increase the convergence speed, some experiments have been performed with various values of λ and the overrelaxation factor ω. These experiments showed that for this problem $\lambda = 0.99$ and $\omega = 1.6$ might be a good choice. However, the solution time for the overrelaxation process is large compared to that of subroutine SOLVE (LU-decomposition). So an improvement of this part of the program might be possible.

As time discretization the modified Crank-Nicolson scheme is used ($\theta = \frac{1}{2}$), combined with a Newton linearization.

Remark

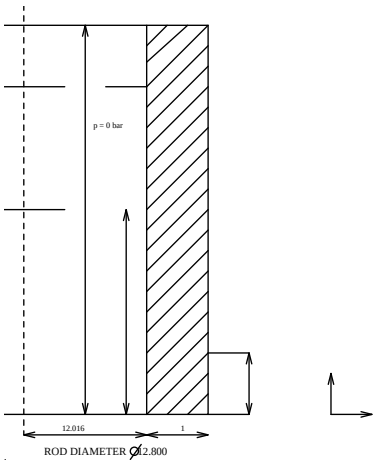


Figure 4.2.1.1: Axisymmetric model used for the calculation

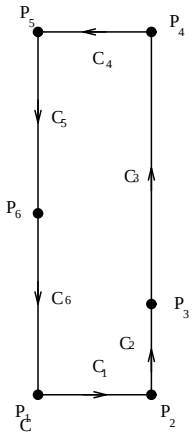


Figure 4.2.1.2: Definition of user points, curves and surface

Since the number of degrees of freedom at curve C6 (3) is unequal to the number of degrees of freedom at the internal elements (2), line elements must be introduced at curves C6 and C1. At curve C2, both line elements and boundary elements may be used. In the program line elements have been chosen, but the results with boundary elements are exactly the same.

To get this example in your local directory type:

```
sepgetex pump
```

To run it perform the following steps:

```
sepmesh pump.msh
view mesh
seplink pump
pump < pump.prb
seppost pump.pst
```

The mesh is generated by the mesh generator SEPMESH.

The region is subdivided into triangles by the submesh generator "RECTANGLE". As an example linear triangles have been used.
SEPMESH needs an input file. An example of an input file for this region is given below:

```
**************************************************************************
*
*      File:  pump.msh
*
*      Contents:  Input for mesh generation part of example described in
*                 Section 4.2.1 in the manual standard problems
*                 Elastohydrodynamic lubrication of an oil pumping ring seal
*                 Submesh generator RECTANGLE is used
**************************************************************************
*
mesh2d
   points

      p1=(6.008d3,0d0)
      p2=(7.008d3,0d0)
      p3=(7.008d3,1.5d3)
      p4=(7.008d3,8d3)
      p5=(6.008d3,8d3)
      p6=(6.008d3,3.5d3)
   curves
      c1=line1(p1,p2,nelm=2)
      c2=line1(p2,p3,nelm=3)
      c3=line1(p3,p4,nelm=13)
      c4=line1(p4,p5,nelm=2)
      c5=line1(p5,p6,nelm=9)
      c6=line1(p6,p1,nelm=7)
   surfaces
      s1=rectangle3(n=2,m=16,c1,c2,c3,c4,c5,c6)
   meshline
*
*  introduction of line elements
*
*  line elements are necessary because the number of degrees of freedom
*  at curve c6 (3) differs from that in the internal elements (2)
*  The line elements at c2 may be replaced by boundary elements, the other
*  ones, however are necessary
*
      lelm1=(shape=1,c6)
      lelm2=(shape=1,c1)
      lelm3=(shape=1,c2)
   meshsurf
*
*  Only one surface element
*
      selm4=(s1)
   plot
end
```

Mark that the unit used in this mesh is μm instead of m.

Figure 4.2.1.3 shows the mesh generated by SEPMESH.

The internal elements are defined by type number 325. Only the coefficients 6, 7 and 11 have to be

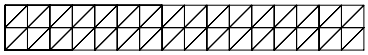


Figure 4.2.1.3: Plot of mesh generated by SEPMESH

defined; all others are zero.
The boundary conditions at sides C5 and C6 are essential boundary conditions, the boundary conditions at sides C2 and C3 are natural boundary conditions requiring no boundary elements at all.

In this particular example, where the problem is time-dependent the complete program is defined. Hence in this case SEPCOMP is not used.
The listing for this program is given by:

```
! ********************************************************************
!
! *   Solution of the elastohydrodynamic lubrication of an oil pumping
! *   ring seal by SEPRAN
! *
! *   Time-dependent problem
! *
! *   At t=0 the non-linear stationary equations are solved by a Newton
! *   iteration
! *   For t>0 a modified Crank-Nicolson scheme is used.
! *   The non-negative pressure condition is imposed by subroutine OVERCS
! *
! *
! *   Programmers:  Kees Kassels and Guus Segal
! *   version 2.0   date   05-12-93
! *
! ********************************************************************
!
      program pump

! ********************************************************************
!
!                       DECLARATIONS

      integer lnmesh, lnprob
      parameter ( lnmesh=100, lnprob = 500 )
      integer kmesh(lnmesh), kprob(lnprob), intmat(5), isol(5),
     +        islold(5), iinstr(3), iincrt(1), iinsol(4), istep, nstep,
     +        iinvec(2), jmetod, istop, iinout(1)
      double precision pi, omeg, rinvec(3)
! ********************************************************************
!
!                       COMMON BOCKS

      include 'SPcommon/ctimen'

! ********************************************************************
!
```

```
!     --- start sepran

      kmesh = 0
      kprob = 0

      kmesh(1) = lnmesh
      kprob(1) = lnprob
      iinstr(1) = 2
      iinstr(2) = 1
      call sepstn ( kmesh, kprob, intmat, iinstr )

!     --- create start vector

      t0 = 0d0
      t = t0
      iincrt(1) = 0
      call creatn ( iincrt, kmesh, kprob, isol )

!     --- non-linear iteration to find solution at t = 0

      iinsol(1) = 0
      call nlnprb ( kmesh, kprob, intmat, isol, iinsol )
      iinout(1) = 0
      call outsol ( kmesh, kprob, isol, iinout, t )

!     --- Compute time-dependent solution with non-constant velocity
!         Set time parameters ( two strokes will be computed )

      omeg  = 151.84d0
      pi    = 4d0*atan(1d0)
      tend  = 4d0*pi/omeg
      nstep = 80
      theta = 0.5d0
      tstep = (tend-t0)/nstep

!     --- define type of matrix for subroutine OVERCS (jmetod=9)

      jmetod = 9
      call commat ( jmetod, kmesh, kprob, intmat )

!     --- Time iteration by Crank-Nicolson (theta=.5)
!         nstep time steps are carried out

      do istep = 1, nstep

!        --- Copy old solution in islold

         call copyvc ( isol, islold )

!        --- Compute u(t n+1/2)

         t = t+0.5d0*tstep
         iinsol(1) = 4
         iinsol(2) = 0
         iinsol(3) = 2
```

```
         iinsol(4) = 2
         call linprb ( kmesh, kprob, intmat, isol, iinsol )

!        --- u(n+1) := 2 u(n+1/2) - u(n)

         iinvec(1) = 2
         iinvec(2) = 39
         rinvec(1) = 2d0
         rinvec(2) = -1d0
         rinvec(3) = 0d0
         call manvec ( iinvec, rinvec, isol, islold, isol, kmesh,
     +                 kprob )
         t = t+0.5d0*tstep

!        --- Output of the solution at 10, 20, 30, ... ,nstep steps

         if ( mod(istep,10).eq.0 ) call outsol ( kmesh, kprob, isol,
     +      iinout, t )

      end do

!     --- Stop SEPRAN

      istop=0
      call finish ( istop )
      end

!     --- funccf for the computation of the velocity as function of time

      function funccf ( ichois, r, z, dummy )
      double precision funccf, r, z, dummy
      integer ichois

      include 'SPcommon/ctimen'

      funccf = 3.492d6 * sin ( 151.84d0 * t )
      end


!     --- function func, for the computation of the starting pressure

      function func ( ichois, r, z, dummy )
      double precision func, r, z, dummy
      integer ichois
      func=max(0d0,6d0*(3.5d3-z)/3.5d3)
      end
```

The following input file may be used to solve the problem:

```
*************************************************************************
*
*      File:  pump.prb
*
*      Contents:  Input for computational part of example described in
*                 Section 4.2.1 in the manual standard problems
```

```
*                 Elastohydrodynamic lubrication of an oil pumping ring seal
*                 SEPMESH must have been run before with input: pump.msh
*                 Program puump must have been linked by seplink
*      Usage:     pump < pump.prb > pump.out
*************************************************************************
constants

   vector_names
      disp_pressure

end
*
*  problem definition
*
problem
*
*    type numbers to be used are:
*       curve c5: 302  (elasto-hydrodynamic Reynolds element)
*       curve c1: 251  (non-zero load for axisymmetric stress analysis)
*       curve c2: 251  (non-zero load for axisymmetric stress analysis)
*       surface s1: 250  (axisymmetric stress analysis)
   types
      elgrp1,(type=302)
      elgrp2,(type=251)
      elgrp3,(type=251)
      elgrp4,(type=250)
*
*     essential boundary conditions
*     The pressure is prescribed in user points p1 and p6
*     The displacement is given at curve c4
*
   essbouncond
      degfd3=points(p1)
      degfd3=points(p6)
      degfd1,degfd2=curves (c4)
end

*  Structure of matrix

matrix
end

*  Creation of start vector (displacement zero)

create
   degfd3, func = 1                     # pressure given by func
   user point (p1), degfd3, value = 6   # prescribed boundary condition
end

*  Input for non-linear solver

nonlinear_equations
   global_options, maxiter = 10, accuracy=1d-3, lin_solver=1
   equation 1
      fill_coefficients = 1
```

```
end

*  Coefficients for non-linear start

coefficients, sequence_number = 1
   elgrp 1 (nparm=6)                 # Elastic Reynolds element
      coef 1 = 12d3                  # Diameter of rod
      coef 2 = 0.0278d-6             # Dynamic viscosity
      coef 3 = 0                     # Constant k
      coef 4 = 0                     # Reference pressure
      coef 5 = 0                     # Velocity
      icoef 6 = 2                    # Stationary, Newton linearization
   elgrp 2 (nparm=25)                # External boundary load
      icoef 2 = 2                    # Axisymmetric stress
      coef 6 = 0                     # Tr
      coef 7 = 6                     # Tz
   elgrp 3 (nparm=25)                # External boundary load
      icoef 2 = 2                    # Axisymmetric stress
      coef 6 = -11                   # Tr
      coef 7 = 0                     # Tz
   elgrp 4 (nparm=45)                # Elasticity element
      icoef 2 = 2                    # Axisymmetric stress
      coef 6 = 5.27d4                # E
      coef 7 = 0.44                  # nu
end

solve, sequence_number = 1           # Direct solver for non-linear problem
end

* Input for linear time-dependent problem

coefficients, sequence_number = 2
   elgrp 1 (nparm=6)                 # Elastic Reynolds element
      coef 1 = 12d3                  # Diameter of rod
      coef 2 = 0.0278d-6             # Dynamic viscosity
      coef 3 = 0                     # Constant k
      coef 4 = 0                     # Reference pressure
      coef 5 = func=1                # Velocity
      icoef 6 = 4                    # Instationary, Newton linearization
   elgrp 2 (nparm=25)                # External boundary load
      icoef 2 = 2                    # Axisymmetric stress
      coef 6 = 0                     # Tr
      coef 7 = 6                     # Tz
   elgrp 3 (nparm=25)                # External boundary load
      icoef 2 = 2                    # Axisymmetric stress
      coef 6 = -11                   # Tr
      coef 7 = 0                     # Tz
   elgrp 4 (nparm=45)                # Elasticity element
      icoef 2 = 2                    # Axisymmetric stress
      coef 6 = 5.27d4                # E
      coef 7 = 0.44                  # nu
end


solve, sequence_number = 2           # Iterative solver for linear problem
```

```
  iteration_method = overrelaxation, accuracy = 1d-2, maxiter = 10000//
  niter1 = -2, lambda=.99, omega=1.6, minimum = 0, start=old_solution//
  degfd = 3
end

end_of_sepran_input
```

4.3 Decoupled elasticity-flow interaction for a bearing (Reynolds equation coupled with mechanical elements)

4.3.1 An example of a combined Reynolds-elasticity problem: A hydrostatic thrust bearing on an elastic track

In Section 4.1.5 we calculated the pressure distribution in the lubrication film of a water lubricated, circular thrust bearing sliding on a track. Now, we will examine the same bearing sliding on an elastic track (see Figure 4.3.1.1). The pressure in the lubrication film and the deformation of the track are mutually dependent: The track will deform due to the hydrostatic pressure in the water film, the hydrostatic pressure is dependent on the local film height which is a function of the track deformation. The calculation consists of the iterative solution to 2 sub-problems and the relation

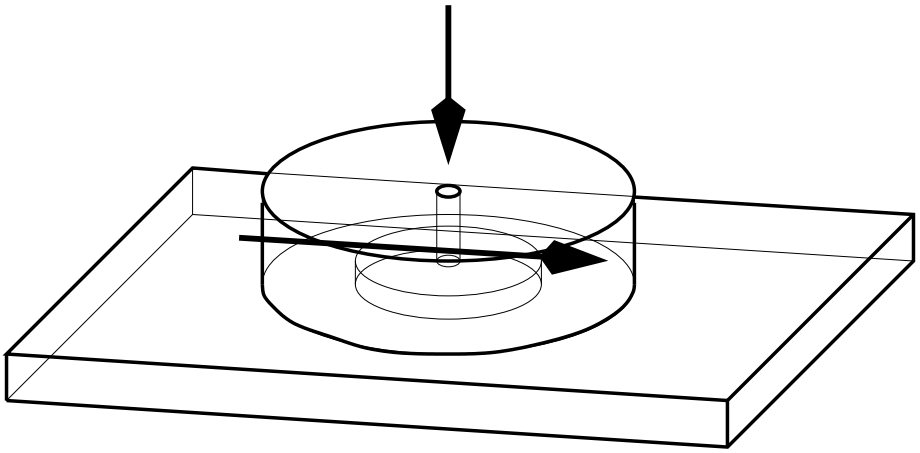


Figure 4.3.1.1: Thrust bearing on an elastic track

between both sub-problems:

- Solution of the Reynolds equation in the lubrication film
- Solution of the elasticity equations in the track
- Calculation of the film height using the track deformation

The solution to the Reynolds equation for this example is similar to the one in Section 4.1.5. Instead of a constant film height a new vector is created with the film height calculated using the following equation:

$$h_i = h_C - (z_i + u_{zi})$$

where h_i is the film height in node i, h_C is the height of the bearing surface above the plane $z = 0$, z_i is the z-co-ordinate of node i and u_{zi} is the displacement in the z-direction of node i. In order to calculate this vector, use is being made of a create vector block with a call to the user subroutine funcvect (see program). Because of to the non-linear supply resistor, the Reynolds equation is solved using an iterative procedure.

The solution to the elasticity equations in the track is performed using the standard elasticity elements (element type 250). The boundary conditions to this problem are:

- Displacement = 0 on the bottom surface of the track.
- Distributed load = -pressure on the Reynolds part of the top surface.

Although the elasticity equation is linear and could be solved using a direct solver, the iterative solution to the combined Reynolds-elasticity problem suggests an iterative approach to the solution of the elasticity equation also.
The solution is assumed to be converged when the change in the calculated film height becomes relatively small.
The parameters for this calculation are presented in table 4.3.1.1. To get this example in your local

Track length	L	1.5	m
Track width	W	1.0	m
Track height	H	0.07	m
Young's modulus track	E	$2.0\ 10^{8}$	N/m^2
Poisson's constant track	v	0.3	$-$
Bearing diameter	D	0.740	m
Waterfilm height	h	$0.1\ 10^{-3}$	m
Recess diameter	D_R	0.530	m
Recess height	h_R	$5.0\ 10^{-3}$	m
Water viscosity	μ	0.001	Ns/m^2
Bearing velocity	u_1	0.25	m/s
Supply resistor	γ	$0.6\ 10^{-6}$	$m^4/N^{1/2}s$
Supply pressure	p_S	$15.0\ 10^{5}$	N/m^2

Table 4.3.1.1: Parameters

directory type:

```
sepgetex bearing4
```

The mesh for this calculation is shown in Figure 4.3.1.2. The pressure along a centerline of the bearing is shown in Figure 4.3.1.3, the displacement along the centerline of the track is shown in Figure 4.3.1.4. The input file for the mesh program (sepmesh) is given here:

```
#
#   Circular Thrust Bearing, 1 recess on a track
#
#   Contents: Mesh file for example 4.3.1
#
#   Author: R.A.J. van Ostayen
#     Date: 23-11-97
#
constants
  reals
    L2 = 0.750      # Half track length
    W2 = 0.500      # Half track width
    H  = -0.070     # Track height
    Rb = 0.370      # Radius of the thrust bearing
    Rr = 0.100      # Radius of the bearing recess

    Cc = 1.0        # Coarse value (centre)
    Cb = 1.0        # Coarse value (bearing)
    Ce = 2.0        # Coarse value (track edge)
  integers
    nz = 2          # number of elements
end

mesh3D
  coarse (unit = 0.05)
```

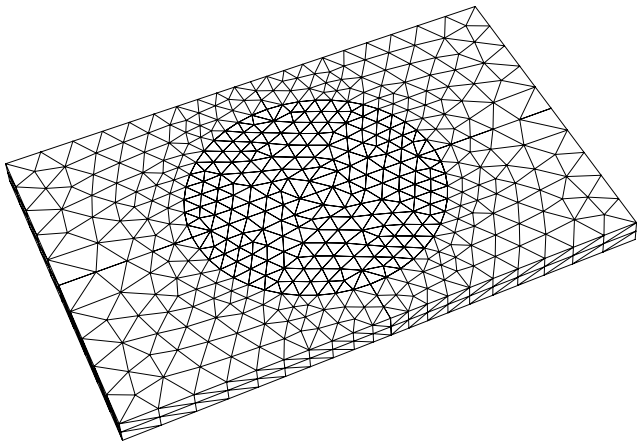


Figure 4.3.1.2: Mesh for the thrust bearing on an elastic track

```
points
  p1  = ( 0.0, 0.0, 0.0,  Cc)

  p2  = (  Rb, 0.0, 0.0,  Cb)
  p3  = ( 0.0,  Rb, 0.0,  Cb)
  p4  = (- Rb, 0.0, 0.0,  Cb)
  p5  = ( 0.0,- Rb, 0.0,  Cb)

  p6  = (  Rr, 0.0, 0.0,  Cb)
  p7  = ( 0.0,  Rr, 0.0,  Cb)
  p8  = (- Rr, 0.0, 0.0,  Cb)
  p9  = ( 0.0,- Rr, 0.0,  Cb)

  p10 = (  L2,  W2, 0.0,  Ce)
  p11 = (- L2,  W2, 0.0,  Ce)
  p12 = (- L2,- W2, 0.0,  Ce)
  p13 = (  L2,- W2, 0.0,  Ce)

  p14 = ( 0.0,  W2, 0.0,  Cb)
  p15 = (- L2, 0.0, 0.0,  Ce)
  p16 = ( 0.0,- W2, 0.0,  Cb)
  p17 = (  L2, 0.0, 0.0,  Ce)

  p18 = ( 0.0, 0.0,   H,  Cc)
```

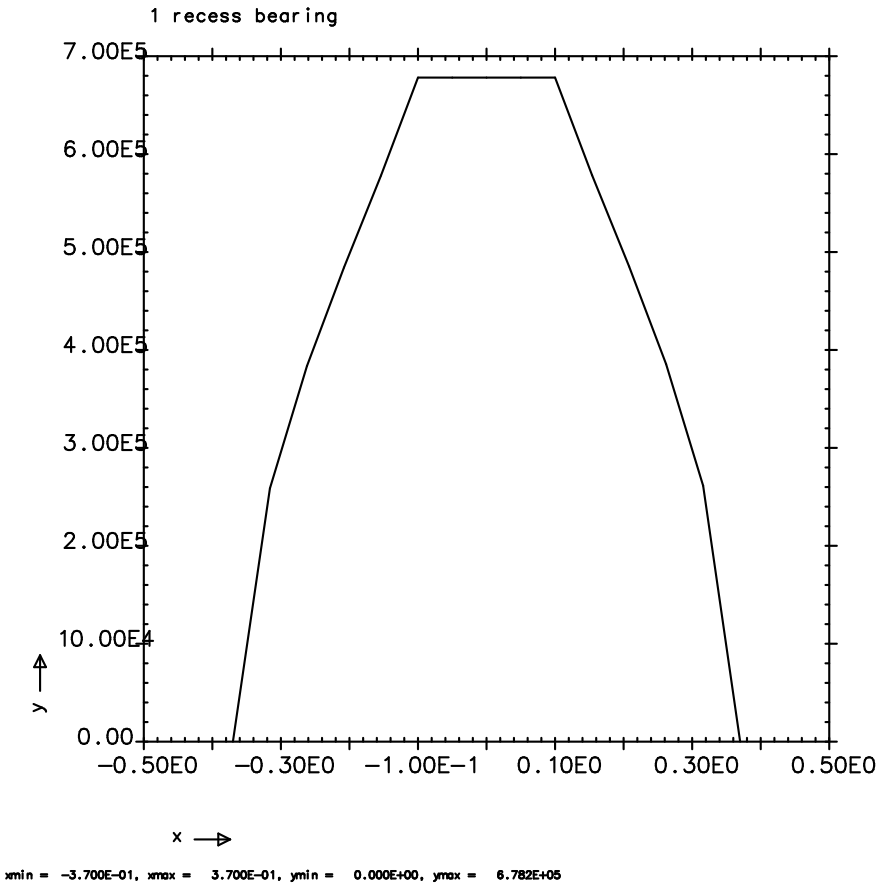


Figure 4.3.1.3: Pressure along a centerline of the bearing

```
  p19 = (  Rb, 0.0,   H,  Cb)
  p20 = ( 0.0,  Rb,   H,  Cb)
  p21 = (- Rb, 0.0,   H,  Cb)
  p22 = ( 0.0,- Rb,   H,  Cb)

  p23 = (  Rr, 0.0,   H,  Cb)
  p24 = ( 0.0,  Rr,   H,  Cb)
  p25 = (- Rr, 0.0,   H,  Cb)
  p26 = ( 0.0,- Rr,   H,  Cb)

  p27 = (  L2,  W2,   H,  Ce)
  p28 = (- L2,  W2,   H,  Ce)
  p29 = (- L2,- W2,   H,  Ce)
  p30 = (  L2,- W2,   H,  Ce)

  p31 = ( 0.0,  W2,   H,  Cb)
  p32 = (- L2, 0.0,   H,  Ce)
  p33 = ( 0.0,- W2,   H,  Cb)
  p34 = (  L2, 0.0,   H,  Ce)
curves
  c1  = carc 1 ( p2 , p3 , p1)
  c2  = carc 1 ( p3 , p4 , p1)
```

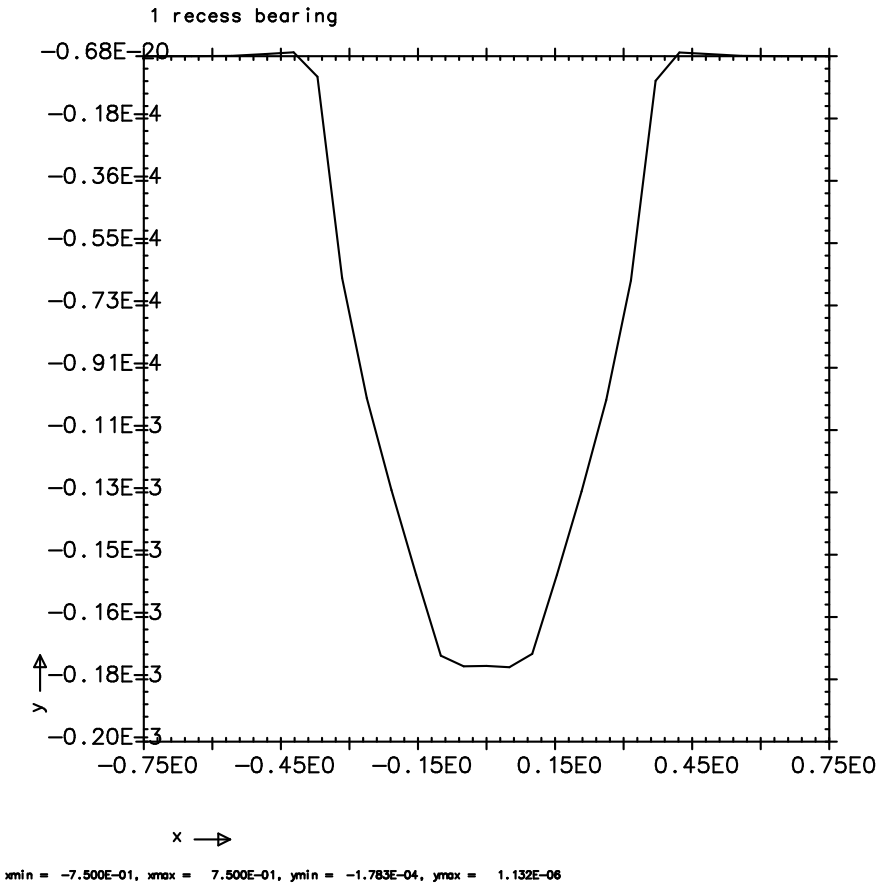


Figure 4.3.1.4: Displacement along the centerline of the track

```
c3  = carc 1 ( p4 , p5 , p1)
c4  = carc 1 ( p5 , p2 , p1)

c5  = carc 1 ( p6 , p7 , p1)
c6  = carc 1 ( p7 , p8 , p1)
c7  = carc 1 ( p8 , p9 , p1)
c8  = carc 1 ( p9 , p6 , p1)

c9  = cline 1 ( p1 , p6)
c10 = cline 1 ( p1 , p7)
c11 = cline 1 ( p1 , p8)
c12 = cline 1 ( p1 , p9)

c13 = cline 1 ( p6 , p2)
c14 = cline 1 ( p7 , p3)
c15 = cline 1 ( p8 , p4)
c16 = cline 1 ( p9 , p5)

c17 = cline 1 ( p2 , p17)
c18 = cline 1 ( p3 , p14)
c19 = cline 1 ( p4 , p15)
c20 = cline 1 ( p5 , p16)
```

```
  c21 = cline 1 ( p10, p14)
  c22 = cline 1 ( p14, p11)
  c23 = cline 1 ( p11, p15)
  c24 = cline 1 ( p15, p12)
  c25 = cline 1 ( p12, p16)
  c26 = cline 1 ( p16, p13)
  c27 = cline 1 ( p13, p17)
  c28 = cline 1 ( p17, p10)

  c29 = translate c1  ( p19, p20)
  c30 = translate c2  ( p20, p21)
  c31 = translate c3  ( p21, p22)
  c32 = translate c4  ( p22, p19)

  c33 = translate c5  ( p23, p24)
  c34 = translate c6  ( p24, p25)
  c35 = translate c7  ( p25, p26)
  c36 = translate c8  ( p26, p23)

  c37 = translate c9  ( p18, p23)
  c38 = translate c10 ( p18, p24)
  c39 = translate c11 ( p18, p25)
  c40 = translate c12 ( p18, p26)

  c41 = translate c13 ( p23, p19)
  c42 = translate c14 ( p24, p20)
  c43 = translate c15 ( p25, p21)
  c44 = translate c16 ( p26, p22)

  c45 = translate c17 ( p19, p34)
  c46 = translate c18 ( p20, p31)
  c47 = translate c19 ( p21, p32)
  c48 = translate c20 ( p22, p33)

  c49 = translate c21 ( p27, p31)
  c50 = translate c22 ( p31, p28)
  c51 = translate c23 ( p28, p32)
  c52 = translate c24 ( p32, p29)
  c53 = translate c25 ( p29, p33)
  c54 = translate c26 ( p33, p30)
  c55 = translate c27 ( p30, p34)
  c56 = translate c28 ( p34, p27)

  c57 = curves ( c21, c22, c23, c24, c25, c26, c27, c28)
  c58 = curves ( c49, c50, c51, c52, c53, c54, c55, c56)
  c59 = line 1 ( p10, p27, nelm =  nz)

  c60 = curves (-c15,-c11, c9 , c13)
  c61 = curves (-c19,-c15,-c11, c9 , c13, c17)
surfaces
  s1  = general 3 ( c13, c1,-c14,-c5 )
  s2  = rotate s1 ( c14, c2,-c15,-c6 )
  s3  = rotate s1 ( c15, c3,-c16,-c7 )
  s4  = rotate s1 ( c16, c4,-c13,-c8 )
```

```
    s5  = general 3 ( c9 , c5,-c10)
    s6  = rotate s5 ( c10, c6,-c11)
    s7  = rotate s5 ( c11, c7,-c12)
    s8  = rotate s5 ( c12, c8,-c9 )

    s9  = general 3 ( c21,-c18,-c1 , c17, c28)
    s10 = similar s9 (-c22,-c18, c2 , c19,-c23)
    s11 = rotate s9 ( c25,-c20,-c3 , c19, c24)
    s12 = similar s9 (-c26,-c20, c4 , c17,-c27)

    s13 = translate s1  ( c41, c29,-c42,-c33)
    s14 = translate s2  ( c42, c30,-c43,-c34)
    s15 = translate s3  ( c43, c31,-c44,-c35)
    s16 = translate s4  ( c44, c32,-c41,-c36)

    s17 = translate s5  ( c37, c33,-c38)
    s18 = translate s6  ( c38, c34,-c39)
    s19 = translate s7  ( c39, c35,-c40)
    s20 = translate s8  ( c40, c36,-c37)

    s21 = translate s9  ( c49,-c46,-c29, c45, c56)
    s22 = translate s10 (-c50,-c46, c30, c47,-c51)
    s23 = translate s11 ( c53,-c48,-c31, c47, c52)
    s24 = translate s12 (-c54,-c48, c32, c45,-c55)

    s25 = surfaces ( s1 , s2 , s3 , s4 , //
                     s5 , s6 , s7 , s8 , //
                     s9 ,-s10, s11, -12)
    s26 = surfaces ( s13, s14, s15, s16, //
                     s17, s18, s19, s20, //
                     s21,-s22, s23,-s24)
    s27 = pipesurface 3 ( c57, c58, c59)
  volumes
    v1  = pipe 11 ( s25, s26, s27)
  meshsurf
    selm1 = s1, s4
    selm2 = s5, s8
  meshvolume
    velm3 = v1
  plot, eyepoint = (-2.0, -4.0, 4.0)
end
```

The input file for the calculation program is given here followed by the source of the program (needed because of the call to funcvect).

```
#
#   Circular Thrust Bearing, 1 recess on a track
#
#   Contents: Problemfile for example 4.3.1
#
#   Author: R.A.J. van Ostayen
#     Date: 23-11-97
#
constants
  reals
```

```
    Hf = 0.1d-3     # film height [m]
    Hr = 5.0d-3     # recess height [m]
    vH2O = 0.001    # viscosity water [Ns/m2]
    U = 0.25        # velocity [m/s]
    Ps = 15.0d5     # supply pressure [N/m2]
    G = 0.6d-6      # resistor value (non-linear)
    EPE = 2.0d8     # Young's modulus [N/m2]
    vPE = 0.3       # -

#  Definition of vectors and scalars

   vector_names
     pressure
     displacement
     film_height
     surf_height  !  height of bearing surface relative to z = 0
   scalars
     max_film_height = -1
     max_rel_error = 1e-2
     act_rel_error = 1e8
     old_max               ! store previous value of max_film_height

end

# Reynolds equation (lubrication problem)
problem 1
  types
    elgrp 1 = (type = 325)
    elgrp 2 = (type = 325)
    elgrp 3 = (type = 0)
  natbouncond
    bngrp 1 = (type = 304)
  bounelements
    belm 1 = points (p6, p7, p8, p9)
  essbouncond
    degfd 1 = curves (c1 to c4)
# elasticity equation
problem 2
  types
    elgrp 1 = (type = 0)
    elgrp 2 = (type = 0)
    elgrp 3 = (type = 250)
  natbouncond
    bngrp 1 = (type = 251)
  bounelements
    belm 1 = surfaces (s1 to s8)
  essbouncond
    degfd 1 = degfd 2 = degfd 3 = surfaces (s26)
end

structure
  # initialize vectors
  create_vector pressure, problem = 1, surfaces (s5 to s8), value = 10e5
  create_vector displacement, problem = 2, value = 0
```

```
  create_vector surf_height, problem = 1, value = Hf

  # while no convergence (actual relative error > max. rel. error)
  while (act_rel_error>max_rel_error ) do

    old_max = max_film_height  ! store present max film height

    # calculate film height and max. film height, at thos moment via input block
    create_vector film_height, problem = 1, sequence_number = 1

    max_film_height, norm = 3, film_height

    act_rel_error = abs(1-max_film_height/old_max)

    print max_film_height, text = 'Max filmheight [m]: '
    print max_rel_error, text = '   Conv. criterion: '
    print act_rel_error, text = '      Conv. number: '

    # solve Reynolds equation
    solve_nonlinear_system pressure, problem = 1, maxiter = 50//
      accuracy = 1e-4, print_level = 0, criterion = relative


    # solve elasticity equations
    solve_linear_system displacement, seq_coef = 2, problem = 2

  end_while

  print max_film_height, text = 'Max filmheight [m]: '
  print max_rel_error, text = '   Conv. criterion: '
  print act_rel_error, text = '      Conv. number: '

  output
end

# matrix (iterative method)
matrix
  storage_scheme = compact, symmetric, problem = 1
  storage_scheme = compact, symmetric, problem = 2
end

# calculate film height
create vector, sequence_number = 1
  value = Hf
  surfaces (s1 to s8), old_vector = 100//
    seq_vectors = (displacement, surf_height)
end

# coefficients for Reynolds equation
coefficients, sequence_number = 1, problem = 1
  elgrp 1 (nparm = 20)
    icoef 1 = 0
    icoef 5 = 1
    coef 6  = old_solution film_height
    coef 7  = vH2O
```

```
    coef 11 = U
    coef 12 = 0
    coef 19 = 0
    coef 20 = 0
  elgrp 2 (nparm = 20)
    icoef 1 = 0
    icoef 5 = 1
    coef 6  = Hr
    coef 7  = vH2O
    coef 11 = U
    coef 12 = 0
    coef 19 = 0
    coef 20 = 0
  bngrp 1 (nparm = 3)
    icoef 1 = 1
    coef 2  = G
    coef 3  = Ps
end

# coefficients for elasticity equation
coefficients, sequence_number = 2, problem = 2
  elgrp 3 (nparm = 45)
    icoef 2 = 0
    coef 6  = EPE
    coef 7  = vPE
  bngrp 1 (nparm = 25)
    icoef 1 = 2
    icoef 2 = 0
    coef 8  = old solution pressure, coef = -1
end

# input for linear solver (Reynolds equation)
solve, sequence_number = 1
  iteration_method = cg, accuracy = 1e-2, start = old_solution, print_level = 0
end

end_of_sepran_input

!    ************************************************************
!    *
!    *  COMPPROGRAM: THRUST BEARING ON ELASTIC TRACK
!    *  usage: comp < in.prb
!    *
!    ************************************************************
!
      program ctbtrk
      call sepcom(0)
      end

      subroutine funcvect( ichoice, ndim, coor, numnodes, uold,
     +                     nuold, result, nphys)

      implicit none
      integer ichoice, ndim, numnodes, nuold, nphys
      double precision coor(ndim,numnodes),
```

```
     +  uold(numnodes,nphys,nuold), result(numnodes,*)

!     coor(1/2/3, *) = node co-ordinates
!     uold(*, 1/2/3, 1) = displacement vector (ex, ey, ez)
!     uold(*, 1, 2) = contact plane height

      integer i
      double precision z_coor, z_disp, h_contact

      if (ichoice .eq. 100) then
        do i = 1, numnodes
          z_coor = coor(3,i)
          z_disp = uold(i,3,1)
          h_contact = uold(i,1,2)
          result(i,1) = h_contact - (z_coor + z_disp)
        end do
      end if

      end
```

The input file for the post-processing program is given here:

```
#
#   Circular Thrust Bearing, 1 recess on elastic track
#
#   Contents: Post-processing file for example 4.3.1
#
#   Author: R.A.J. van Ostayen
#     Date: 23-11-97
#
postprocessing

  plot identification, text = '1 recess bearing', origin = (3,18)
  open plot
    plot boundary function pressure, curves (c60), //
      arc_scales = (-0.37, 0.37), //
      scales = (-0.5, 0.5, 0.0, 7.0d5), steps = (10, 7)
  close plot
  open plot
    plot boundary function displacement, degfd 3, curves (c61), //
      arc_scales = (-0.75, 0.75), //
      scales = (-0.75, 0.75, -2.0d-4, 0.0d0), steps = (10, 11)
  close plot
  open plot
    plot boundary function film_height, curves (c60), //
      arc_scales = (-0.37, 0.37)
  close plot
end
```

5 Mechanical elements

5.1 Linear elastic problems

5.1.1 The hole-in-plate problem (example of plane stress)

Consider the plate in Figure 5.1.1.1

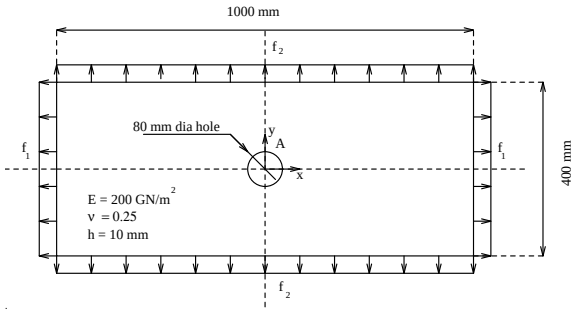


Figure 5.1.1.1: The hole-in-plate problem

For symmetry reasons it is sufficient to discretize only one quarter of the plate. The problem is solved by bilinear quadrilateral elements. For the generation of the mesh we define the 6 user points, and 5 curves. The definition of user points and curves is given in Figure 5.1.1.2.

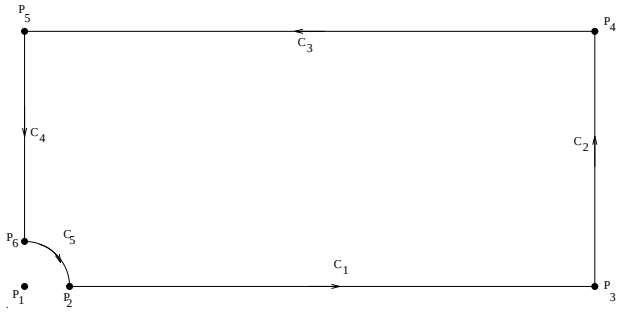


Figure 5.1.1.2: Definition of user points and curves

This example is nearly identical to the one described in the SEPRAN INTRODUCTION Section 7.2. As an example we use quadrilaterals instead of triangles. Consequence is that the mesh generation is somewhat more sensitive to the spacing and therefore a more uniform spacing is used. In order to check the mesh the option CHECK_MESH is used.

The material is supposed to be orthotropic (IGPROB=3).
The following parameters are used:

Young's modulus: $E_1 = 10^7$ N/m^2, $E_2 = 10^7$ N/m^2
Poisson's ratio: $\nu_2 = 0.25$
Plate thickness is 0.01m.

The boundary loads f_1 and f_2 are given by:

$f_1 \ = \ -10^4$ N/m^2 $f_2 \ = \ 0$ N/m^2

Essential boundary conditions:

symmetry axis: C_1: $v = 0$ C_4: $u = 0$

The mesh is created by SEPMESH with the following input file:

```
*****************************************************************************
*
*      File:  plathol1.msh
*
*      Contents:  Input for mesh generation part of the example as described
*                 in the SEPRAN STANDARD PROBLEMS Section 5.1.1
*
*      Usage:    In UNIX:   sepmesh plathol1.msh
*      Usage:    In DOS:    sepmesh plathol1.msh
*
*****************************************************************************
*
*
*
*  mesh for hole in plate problem
*
mesh2d
*  unit length is 1 cm
   coarse(unit=0.01)

*  definition of user points  with corresponding coarseness:

   points
      p1=(0,    0,    1)
      p2=(0.04, 0,    1)
      p3=(0.5,  0,    2)
      p4=(0.5,  0.2,  2)
      p5=(0,    0.2,  1)
      p6=(0,    0.04, 1)

*  curves defining the surfaces:

   curves
      c1=cline1(p2,p3,nodd=3)
      c2=cline1(p3,p4,nodd=3)
      c3=cline1(p4,p5,nodd=3)
      c4=cline1(p5,p6,nodd=3)
      c5=carc1(p6,p2, -p1,nodd=2)

*   the surface is created by general:
*   bilinear quadrilaterals

   surfaces
      s1=general5(c1,c2,c3,c4,c5)

*  plot:
*  the submesh is skipped
*  numbers are not plotted

   plot(jmark=5,numsub=1)

*  Check the mesh:
```

```
   check_level = 2
end
```

Figure 5.1.1.3 shows the mesh created by SEPMESH.

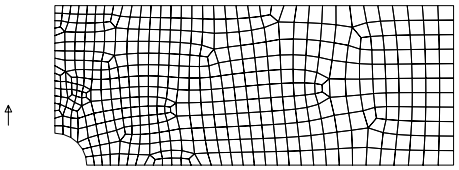


Figure 5.1.1.3: Mesh plot of hole-in-plate region

Once the mesh has been generated, sepcomp may be run to compute the displacement. For the linear elasticity problem element type 250 may be used, see Section 5.1. Type 250 requires 45 coefficients, however, it is sufficient to give only the non-zero values.

```
*****************************************************************************
*
*      File:  plathol4.prb
*
*      Contents:  Input for computational part of the example as described
*                 in the SEPRAN manual STANDARD PROBLEMS Section 3.1.1
*                 Model used: IGPROB=3, i.e. orthotropic material
*
*      Usage:    sepmesh should have been run with input: plathole.msh
*                In UNIX:   sepcomp plathol4.prb > plathol4.out
*                In DOS:    sepcomp plathol4.prb
*
*****************************************************************************
*
*
*
*  problem definition
*
problem

*  only one type is used (250: Linear elastic element)

   types
      elgrp1, (type=250)

*   for the boundary loads natural boundary condition elements are necessary
*   type number: 251 linear line element
```

```
*   Different element groups are used for the curves c2 and c3

   natbouncond
      bngrp1, (type=251)
      bngrp2, (type=251)
   bounelements
      belm1 = curves (shape=1, c2)
      belm2 = curves (shape=1, c3)

*  essential boundary conditions:
*  the curves c1 and c4 are symmetry axis, hence the normal displacements
*  must be suppressed

essbouncond
   degfd2 = curves0(c1)
   degfd1 = curves0(c4)
end
*  Define type of matrix
matrix
   method = 1
end

* Define coefficients

coefficients
   elgrp1 (nparm=45)
     icoef 2 = 3                          # IGPROB=3 (orthotropic material)
     coef  6 = ( value = 1d7  )           # E_1
     coef  7 = ( value = 1d7  )           # E_2
     coef  8 = ( value = 0.3  )           # nu_1
     coef  9 = ( value = 0.3  )           # nu_2
     coef 10 = ( value = 0.384615384d7 )  # G_2
     coef 27 = ( value = 0.01 )           # h
   bngrp1 (nparm=25)
     coef  6 = ( value = -1d4 )           # T_x
     coef  9 = ( value = 0.01 )           # h
   bngrp2 (nparm=25)
     coef  6 = ( value = 0d0 )            # T_x
     coef  9 = ( value = 0.01 )           # h
end

*  The matrix is positive definite

solve
   positive definite
end
output
      v1 = icheld = 6
      v2 = icheld = 7
end
end_of_sepran_input
```

Program seppost allows us to print and plot the solution. It requires input from the standard input file.
If, for example, we want to print the displacements and the stresses, make a vector plot of the

displacements, make a contour plot of the three non-zero components of the stress tensor as well as coloured contour plots, plus some prints at the boundaries then the following input file may be used:

```
************************************************************************
*
*      File:  plathole.pst
*
*      Contents:  Input for post processing part of the example as described
*                 in the manual Standard Problems Section 5.1.1
*
*      Usage:     sepmesh should have been run with input: plathol1.msh
*                 sepcomp should have been run with input: plathol4.prb
*                 In UNIX:   seppost plathole.pst
*
************************************************************************
*
*
postprocessing
   name v0 = displacement
   name v1 = stresses
   print v0
   plot identification, text=' Test example "hole in plate" ', origin = (15,18)
   open plot
      plot vector v0
      plot text, text = 'Displacements vectors', origin = (0.15,-0.04)
   close plot
   print v1
   open plot
      plot contour v1, degfd=1,smoothing factor = 1
      plot text, text = 'Contours of xx-component of stress'//
                 origin = (0.15,-0.04)
   close plot
   open plot
      plot contour v1, degfd=2,smoothing factor = 1
      plot text, text = 'Contours of yy-component of stress'//
                 origin = (0.15,-0.04)
   close plot
   open plot
      plot contour v1, degfd=4,smoothing factor = 1
      plot text, text = 'Contours of xy-component of stress'//
                 origin = (0.15,-0.04)
   close plot
   open plot
      plot coloured contour v1, degfd=1
      plot text, text = 'Contours of xx-component of stress'//
                 origin = (0.15,-0.04)
   close plot
   open plot
      plot coloured contour v1, degfd=2
      plot text, text = 'Contours of yy-component of stress'//
                 origin = (0.15,-0.04)
   close plot
   open plot
      plot coloured contour v1, degfd=4
```

```
      plot text, text = 'Contours of xy-component of stress'//
                 origin = (0.15,-0.04)
   close plot
   print boundary function v0, curves(c1,c2,c3)
   print boundary function v1, curves(c2,c3,c4,c5)
   print boundary function v1, curves(c2), degfd=1
   print boundary function v1, curves(c3), degfd=2
   print boundary function v1, curves(c4), degfd=3
end
```

Figure 5.1.1.4 shows the required vector plot, Figures 5.1.1.5 - 5.1.1.7 the contour plots of the stresses. The coloured plots are not shown in this manual.

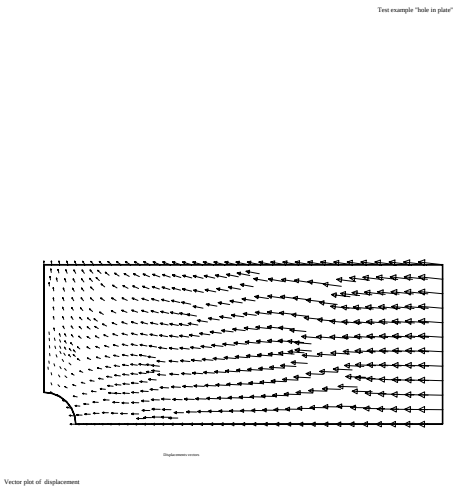


Figure 5.1.1.4: Vector plot of displacements in hole-in-plate problem

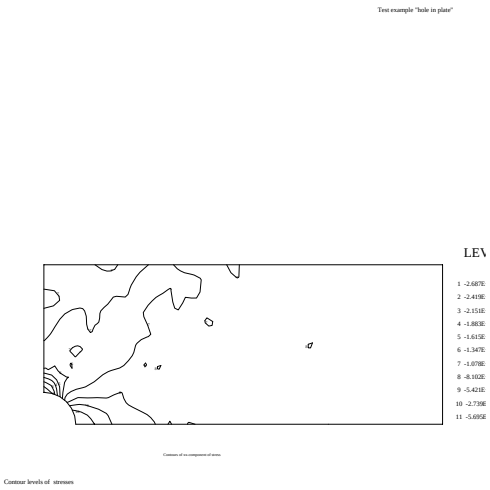


Figure 5.1.1.5: Contour plot of σ_{xx} in hole-in-plate problem

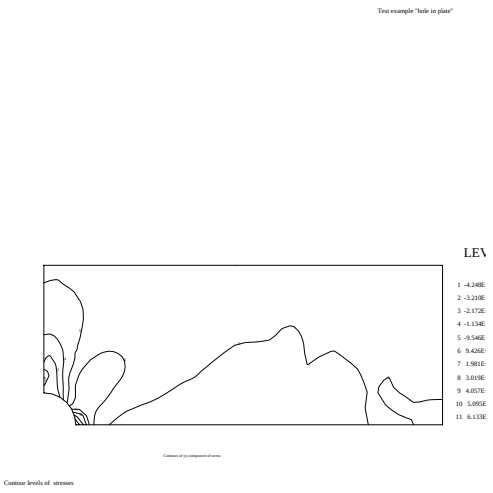


Figure 5.1.1.6: Contour plot of σ_{yy} in hole-in-plate problem

5.1.2 A simple normal load example

In this example we consider some variations on the tube as sketched in Figure 5.1.2.1

In the first two examples the outer boundary of the tube has displacement zero, whereas at the inner circle a normal load of $10^5 N$ is given. The thickness of the pipe is equal to 1 m. This example shows how a normal load could be treated in combination with the stress elements of this chapter. Since the outer side of the tubes has zero displacement, this is an example of plane strain (IGPROB=1). If the outer side could move freely, it would have been a plane stress (IGPROB=0) example. This is the case in the last two examples.
To get these examples into your local directory use:

```
sepgetex normload$
```

$ refers to the sequence number of the example. The available sequence numbers are 1 to 4.
To run such an example carry out the following commands:

```
sepmesh normload1.msh
view mesh
sepcomp normload1.prb
seppost normload1.pst
view results
```

viewing may be done by: sepdisplay, xsepask or xsepplot, like

```
xsepplot sepplot.001
xsepplot sepplot.002 ....
sepview sepplot.001  (provided you have os-motif)
```

output may be redirected to a file by:

```
command > file..
```

normload1 must be replaced by normload2, 3 or 4 depending on the example.
The mesh is generated by program SEPMESH using bilinear quadrilateral elements. The input file for SEPMESH for the first two examples is:

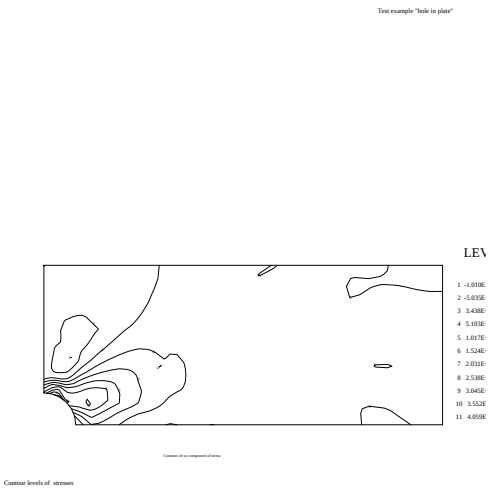


Figure 5.1.1.7: Contour plot of τ_{xy} in hole-in-plate problem

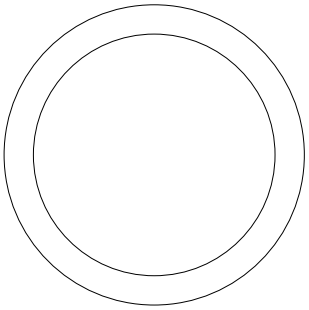


Figure 5.1.2.1: Hollow tube with internal load

```
#  normload1.msh
#
#  mesh for normal load example
#  See Manual Standard Elements Section 5.1.2
#  and examples manual Section 5.1.2
#
#  To run this file use:
#     sepmesh normload1.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      nelw =  5         # number of elements in wall-thickness
      nelc = 40         # number of elements in circumference direction
   reals
      ri     = 2         # inner radius
      ro     = 3         # outer radius
end
```

```
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1 = (0,0)      # centre of circles
      p2 = ( ri,0)    # point at inner circle
      p3 = ( ro,0)    # point at outer circle
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1 = arc1(p2,p2,p1,nelm= nelc)  # inner circle
      c2 = arc1(p3,p3,p1,nelm= nelc)  # outer circle
      c3 = line1(p2,p3,nelm= nelw)    # connection line, only necessary to
                                      # define a closed region
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1 = rectangle 5 ( c1,c3,-c2,-c3 )  # number of elements at opposite
                                          # sides is constant
                                          # See Users Manual Section 2.4.2
   plot                                   # make a plot of all parts
                                          # and also of the final mesh
                                          # See Users Manual Section 2.2
end
```

Figure 5.1.2.2 shows the mesh created by SEPMESH.

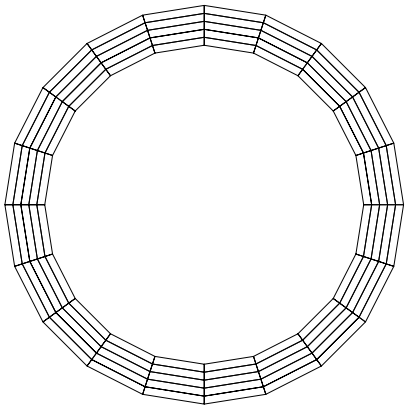


Figure 5.1.2.2: Mesh plot of normal load example

Once the mesh has been generated, sepcomp may be run to compute the displacement. For the linear elasticity problem element type 250 may be used, see Section 5.1. Type 250 requires 45 coefficients, however, it is sufficient to give only the non-zero values.
The physical parameters used are:
Young's modulus: $E = 10^7$ N/m^2
Poisson's ratio: $\nu = 0.3$
Plate thickness is 1m.

In order to prescribe the normal load we may choose between local transformations in combination with ILOAD=1, or no transformation and ILOAD=4. Both give exactly the same results. We give the input files in both cases:

```
#  normload1.prb
#  Problem definition for normal load example
#  See Manual Standard Elements Section 5.1.2
#  and examples manual Section 5.1.2
#
#  To run this file use:
#     sepcomp normload1.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#  Example with local transform
#
constants
   vector_names
      displacement
end
#  Define the type of problem to be solved
problem                      # See Users Manual Section 3.2.2
   types                     # Define types of elements,
                             # See Users Manual Section 3.2.2
      elgrp1 (type=250)      # Linear elastic element
                             # See Manual Standard Elements Section 5.1
   natboundcond              # Define type of natural boundary conditions (loads)
                             # See Users Manual Section 3.2.2
      bngrp1 (type=251)      # Given load for linear elastic element
                             # See Manual Standard Elements Section 5.1
   bounelements              # Define where the natural boundary conditions
                             # are given. See Users Manual Section 3.2.2
      belm1=curves(c1)       # Load at inner circle c1
   essbouncond               # Define where essential boundary conditions are
                             # given (not the value)
                             # See Users Manual Section 3.2.2
      curves(c2)             # Displacement is prescribed on outer circle
   localtransform            # Define local transformation
                             # See Users Manual Section 3.2.2
      curves(c1)             # The first unknown on curve 1 is in the normal
                             # direction, the second one in the tangential
                             # direction
end

#  Define the structure of the large matrix

matrix                       # See Users Manual Section 3.2.4
```

```
   symmetric                # The matrix is symmetrical and stored as profile
                            # matrix, hence a direct solver is applied
end

#  Define the coefficients for the problem

coefficients                # See Users Manual Section 3.2.6
   elgrp1 (nparm=45)        # The number of coefficients for type 250 is 45
                            # See Manual Standard Elements Section 5.1
      icoef2 = 1            # Plane strain
      coef 6 = 1d7          # Youngs modulus E
      coef 7 = 0.3          # Poisson ratio nu
   bngrp1 (nparm=25)        # The number of coefficients for type 251 is 25
                            # See Manual Standard Elements Section 5.1
      icoef1 = 1            # ILOAD = 1, load in co-ordinate direction
      icoef2 = 1            # Plane strain
      coef 6 = -1d5         # Load in the first co-ordinate direction
                            # Due to the local transformation, this is
                            # the normal direction
end

# The following input parts are not explicitly given:
#
# essential boundary conditions, See Users Manual Section 3.2.5
#                               Reason, the given displacement is 0
# solve, See Users Manual Section 3.2.8
#                               Reason, the default solver is used
# output, See Users Manual Section 3.2.13
#                               Reason, the default output is written
end_of_sepran_input
```

```
#  normload2.prb
#  Problem definition for normal load example
#  See Manual Standard Elements Section 5.1.2
#  and examples manual Section 5.1.2
#
#  To run this file use:
#     sepcomp normload2.prb
#
#  Reads the file meshoutput
#  Creates the files sepcomp.inf and sepcomp.out
#
#  Example with ILOAD=4
constants
   vector_names
      displacement
end
#
#  Define the type of problem to be solved
problem                     # See Users Manual Section 3.2.2
   types                    # Define types of elements,
                            # See Users Manual Section 3.2.2
```

```
      elgrp1 (type=250)     # Linear elastic element
                            # See Manual Standard Elements Section 5.1
   natboundcond             # Define type of natural boundary conditions (loads)
                            # See Users Manual Section 3.2.2
      bngrp1 (type=251)     # Given load for linear elastic element
                            # See Manual Standard Elements Section 5.1
   bounelements             # Define where the natural boundary conditions
                            # are given. See Users Manual Section 3.2.2
      belm1=curves(c1)      # Load at inner circle c1
   essbouncond              # Define where essential boundary conditions are
                            # given (not the value)
                            # See Users Manual Section 3.2.2
      curves(c2)            # Displacement is prescribed on outer circle
end

#  Define the structure of the large matrix

matrix                      # See Users Manual Section 3.2.4
   symmetric     # Symmetrical profile matrix
                            # matrix, hence a direct solver is applied
end

#  Define the coefficients for the problem

coefficients                # See Users Manual Section 3.2.6
   elgrp1 (nparm=45)        # The number of coefficients for type 250 is 45
                            # See Manual Standard Elements Section 5.1
      icoef2 = 1            # Plane strain
      coef 6 = 1d7          # Youngs modulus E
      coef 7 = 0.3          # Poisson ratio nu
   bngrp1 (nparm=25)        # The number of coefficients for type 251 is 25
                            # See Manual Standard Elements Section 5.1
      icoef1 = 4            # ILOAD = 4, load in normal direction
      icoef2 = 1            # Plane strain
      coef 6 = -1d5         # Load in the normal direction
end

# The following input parts are not explicitly given:
#
# essential boundary conditions, See Users Manual Section 3.2.5
#                               Reason, the given displacement is 0
# solve, See Users Manual Section 3.2.8
#                               Reason, the default solver is used
# output, See Users Manual Section 3.2.13
#                               Reason, the default output is written
end_of_sepran_input
```

Program seppost allows us to print and plot the solution. It requires input from the standard input file.
A very simple example is given in the following file:

```
# normload1.pst
#  Input file for postprocessing for normal load example
#  See Manual Standard Elements Section 5.1.2
#  and examples manual Section 5.1.2
```

```
#
#  To run this file use:
#     seppost normload1.pst > normload1.out
#
#  Reads the files meshoutput, sepcomp.inf and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2
   print displacement             # Print the complete displacement
                                  # See Users Manual Section 5.3
   print displacement, curves=c1  # Print the displacement along the inner
                                  # circle
                                  # See Users Manual Section 5.3
   plot vector displacement       # Make a vector plot of the displacement
                                  # See Users Manual Section 5.4
end
```

In the third example we use only a quarter of the region and make use of the symmetry of the solution. Only the part in the first quadrant is used. Furthermore we do not prescribe the displacement on the outer circle. As a consequence we use plain stress instead of plane strain.
The mesh input file is given by:

```
#  normload3.msh
#
#  mesh for normal load example
#  In this case only a quarter of the region is used
#  See Manual Standard Elements Section 5.1.2
#  and examples manual Section 5.1.2
#
#  To run this file use:
#     sepmesh normload3.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      nelw =  5         # number of elements in wall-thickness
      nelc = 10         # number of elements in circumference direction
   reals
      ri     = 2         # inner radius
      ro     = 3         # outer radius
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1  = (0,0)      # centre of circles
      pd2 = ( ri,0)    # point at inner circle (at 0 degrees)
                       # coordinates are given in radius and angle
      pd3 = ( ri,90)   # point at inner circle (at 90 degrees)
```

```
      pd4 = ( ro,90)   # point at outer circle (at 90 degrees)
      pd5 = ( ro,0)    # point at outer circle (at 0 degrees)
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1 = arc1(p2,p3,p1,nelm= nelc)  # inner circle
      c2 = line1(p3,p4,nelm= nelw)    # line at 90 degrees
      c3 = arc1(p4,p5,-p1,nelm= nelc) # outer circle
      c4 = line1(p5,p2,nelm= nelw)    # line at 0 degrees
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1 = rectangle 5 ( c1,c2,c3,c4 )  # number of elements at opposite
                                        # sides is constant
                                        # See Users Manual Section 2.4.2
   plot                                 # make a plot of all parts
                                        # and also of the final mesh
                                        # See Users Manual Section 2.2
end
```

The corresponding problem input file becomes:

```
#  normload3.prb
#  Problem definition for normal load example
#  In this case only a quarter of the region is used
#  See Manual Standard Elements Section 5.1.2
#  and examples manual Section 5.1.2
#
#  To run this file use:
#     sepcomp normload3.prb
#
#  Reads the file meshoutput
#  Creates the files sepcomp.inf and sepcomp.out
#
#  Example with ILOAD=4 and free outer circle
constants
   vector_names
      displacement
end
#
#  Define the type of problem to be solved
problem                     # See Users Manual Section 3.2.2
   types                    # Define types of elements,
                            # See Users Manual Section 3.2.2
      elgrp1 (type=250)     # Linear elastic element
                            # See Manual Standard Elements Section 5.1
   natboundcond             # Define type of natural boundary conditions (loads)
                            # See Users Manual Section 3.2.2
      bngrp1 (type=251)     # Given load for linear elastic element
                            # See Manual Standard Elements Section 5.1
   bounelements             # Define where the natural boundary conditions
                            # are given. See Users Manual Section 3.2.2
      belm1=curves(c1)      # Load at inner circle c1
```

```
   essbouncond              # Define where essential boundary conditions are
                            # given (not the value)
                            # See Users Manual Section 3.2.2
      degfd1 = curves(c2)   # x-displacement is prescribed on line at 90 degrees
      degfd2 = curves(c4)   # y-displacement is prescribed on line at 0 degrees
                            # The last two are symmetry conditions
end

#  Define the structure of the large matrix

matrix                      # See Users Manual Section 3.2.4
   symmetric                # The matrix is symmetrical and stored as profile
                            # matrix, hence a direct solver is applied
end

#  Define the coefficients for the problem

coefficients                # See Users Manual Section 3.2.6
   elgrp1 (nparm=45)        # The number of coefficients for type 250 is 45
                            # See Manual Standard Elements Section 5.1
      icoef2 = 0            # Plane stress
      coef 6 = 1d7          # Youngs modulus E
      coef 7 = 0.3          # Poisson ratio nu
   bngrp1 (nparm=25)        # The number of coefficients for type 251 is 25
                            # See Manual Standard Elements Section 5.1
      icoef1 = 4            # ILOAD = 4, load in normal direction
      icoef2 = 0            # Plane stress
      coef 6 = -1d5         # Load in the normal direction
end

# The following input parts are not explicitly given:
#
# essential boundary conditions, See Users Manual Section 3.2.5
#                               Reason, the given displacement is 0
# solve, See Users Manual Section 3.2.8
#                               Reason, the default solver is used
# output, See Users Manual Section 3.2.13
#                               Reason, the default output is written
end_of_sepran_input
```

Finally we extend the third example to R^3. To that end the mesh is extended in the third direction. The mesh input file in this case is given by:

```
#  normload4.msh
#
#  mesh for normal load example
#  3D example
#  See Manual Standard Elements Section 5.1.2
#  and examples manual Section 5.1.2
#
#  To run this file use:
#     sepmesh normload4.msh
#
#  Creates the file meshoutput
#
```

```
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      nelw =  5         # number of elements in wall-thickness
      nelc =  5         # number of elements in circumference direction
      nelz =  5         # number of elements in z direction
   reals
      ri     = 2         # inner radius
      ro     = 3         # outer radius
      height = 3         # outer radius
end
#
#  Define the mesh
#
mesh3d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1  = (0,0,0)          # centre of circles
      pd2 = ( ri,0,0)        # point at inner circle (at 0 degrees)
                             # coordinates are given in radius and angle
      pd3 = ( ri,90,0)       # point at inner circle (at 90 degrees)
      pd4 = ( ro,90,0)       # point at outer circle (at 90 degrees)
      pd5 = ( ro,0,0)        # point at outer circle (at 0 degrees)
      p6  = ( ri,0, height)  # point above p2
      p9  = (0,0,0)          # The points p7-p9 are generated by translate
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1 = arc1(p2,p3,p1,nelm= nelc)  # inner circle
      c2 = line1(p3,p4,nelm= nelw)    # line at 90 degrees
      c3 = arc1(p4,p5,-p1,nelm= nelc) # outer circle
      c4 = line1(p5,p2,nelm= nelw)    # line at 0 degrees
      c5 = translate c1 (p6,p7)       # inner circle on upper surface
      c6 = translate c2 (p7,p8)       # line at 90 degrees on upper surface
      c7 = translate c3 (p8,p9)       # outer circle on upper surface
      c8 = translate c4 (p9,p6)       # line at 0 degrees on upper surface
      c9 = line1 (p2,p6,nelm= nelz)   # generating curve for pipe surface
      c10= translate c9 (p3,p7)       # generating curve for pipe surface
      c11= translate c9 (p4,p8)       # generating curve for pipe surface
      c12= translate c9 (p5,p9)       # generating curve for pipe surface
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1 = rectangle 5 ( c1,c2,c3,c4 )  # lower surface
                                        # See Users Manual Section 2.4.2
      s2 = translate s1 ( c5,c6,c7,c8)  # upper surface
                                        # See Users Manual Section 2.4
      s3 = pipesurface 5 ( c1,c5,c9,c10)  # pipe surface along inner curves
                                          # See Users Manual Section 2.4.5
      s4 = pipesurface 5 ( c2,c6,c10,c11) # pipe surface along curves at 90 deg
```

```
      s5 = pipesurface 5 ( c3,c7,c11,c12) # pipe surface along outer curves
      s6 = pipesurface 5 ( c4,c8,c12,c9)  # pipe surface along curves at 0 deg
      s7 = ordered surfaces ( s3, s4, s5, s6 ) # Complete pipe surface
                                               # Necessary for pipe
                                               # See Users Manual Section 2.4
#
# volumes
#
   volumes         # See Users Manual Section 2.5
      v1 = pipe 13 ( s1, s2, s7 )  # Complete region
                                   # See Users Manual Section 2.5.2
   plot(eyepoint(0,-10,10) # make a plot of all parts
                           # and also of the final mesh
                           # See Users Manual Section 2.2
                           # Only with the eye point the complete mesh is drawn
end
```

The mesh is shown in Figure 5.1.2.3

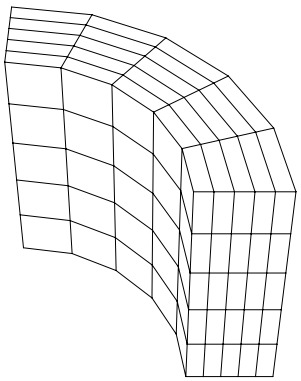


Figure 5.1.2.3: Mesh plot of 3D normal load example

In this case it is necessary to prescribe the z-displacement in at least one point. To make things simple the z-displacement in the upper surface is made equal to zero. The corresponding problem input file is:

```
#  normload4.prb
#  Problem definition for normal load example
#  In this case only a quarter of the region is used
#  See Manual Standard Elements Section 5.1.2
#  and examples manual Section 5.1.2
#
#  To run this file use:
#     sepcomp normload4.prb
#
#  Reads the file meshoutput
```

```
#  Creates the files sepcomp.inf and sepcomp.out
#
#  Example with ILOAD=4 and free outer circle
constants
   vector_names
      displacement
end
#
#  Define the type of problem to be solved
problem                      # See Users Manual Section 3.2.2
   types                     # Define types of elements,
                             # See Users Manual Section 3.2.2
      elgrp1 (type=250)      # Linear elastic element
                             # See Manual Standard Elements Section 5.1
   natboundcond              # Define type of natural boundary conditions (loads)
                             # See Users Manual Section 3.2.2
      bngrp1 (type=251)      # Given load for linear elastic element
                             # See Manual Standard Elements Section 5.1
   bounelements              # Define where the natural boundary conditions
                             # are given. See Users Manual Section 3.2.2
      belm1=surfaces(s3)     # Load at inner circle s3
   essbouncond               # Define where essential boundary conditions are
                             # given (not the value)
                             # See Users Manual Section 3.2.2
      degfd1 = surfaces(s4) # x-displacement is prescribed on line at 90 degrees
      degfd2 = surfaces(s6) # y-displacement is prescribed on line at 0 degrees
      degfd3 = surfaces(s2) # z-displacement is prescribed on upper surface
                             # The last three are symmetry conditions
end

#  Define the structure of the large matrix

matrix                       # See Users Manual Section 3.2.4
   symmetric                 # The matrix is symmetrical and stored as profile
                             # matrix, hence a direct solver is applied
end

#  Define the coefficients for the problem

coefficients                 # See Users Manual Section 3.2.6
   elgrp1 (nparm=45)         # The number of coefficients for type 250 is 45
                             # See Manual Standard Elements Section 5.1
      icoef2 = 0             # Linear Elasticity (3D)
      coef 6 = 1d7           # Youngs modulus E
      coef 7 = 0.3           # Poisson ratio nu
   bngrp1 (nparm=25)         # The number of coefficients for type 251 is 25
                             # See Manual Standard Elements Section 5.1
      icoef1 = 4             # ILOAD = 4, load in normal direction
      icoef2 = 0             # Linear Elasticity (3D)
      coef 6 = -1d5          # Load in the normal direction
end

# The following input parts are not explicitly given:
#
# essential boundary conditions, See Users Manual Section 3.2.5
```

```
#                                Reason, the given displacement is 0
# solve, See Users Manual Section 3.2.8
#                                Reason, the default solver is used
# output, See Users Manual Section 3.2.13
#                                Reason, the default output is written
end_of_sepran_input
```

5.1.3 Time-dependent linear beam response

In this example a simple clamped beam is excited by a time-dependent distributed load (Figure 5.1.3.1), and then released. The beam is clamped in the right-hand side, and the load is applied on top of the beam.
To get this example into your local directory use:

```
sepgetex beamresponse
```

To run such the example carry out the following commands:

```
sepmesh beamresponse.msh
view mesh
seplink beamresponse
beamresponse < beamresponse.prb
seppost beamresponse.pst
view results
```

In Figure 5.1.3.2 the corresponding finite element mesh with linear triangular elements is displayed. For the description of the material of the beam, a linear constitutive law is used in combination with linear geometric assumptions (element type 250). For the time integration the Newmark time integration method is used, with $\beta = 0.25$ and $\gamma = 0.5$. For detailed information about solid time integration, see the Sepran Theory Manual Section 5.6. The beam is loaded by a distributed load $f(t)$ that is equal to:

$$\begin{aligned} f(t) &= 10t \qquad \text{for } 0 \leq t \leq 2.5 \\ f(t) &= 0 \qquad \text{for } t > 2.5 \end{aligned}$$

Due to this load, the beam is going to oscillate. The deformation in x and y direction of the upper left point of the beam is plotted in Figure 5.1.3.3. Because linear geometric assumptions are used, the response is only accurate for small displacements. For large displacements the updated Lagrange formulation of the solid is recommended (element types 200-202).

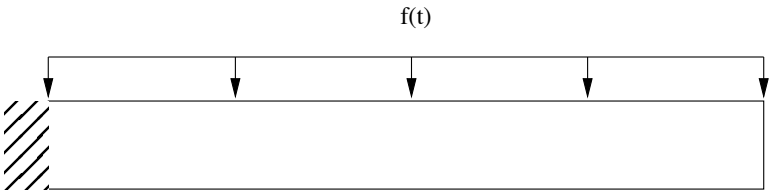


Figure 5.1.3.1: Clamped beam loading

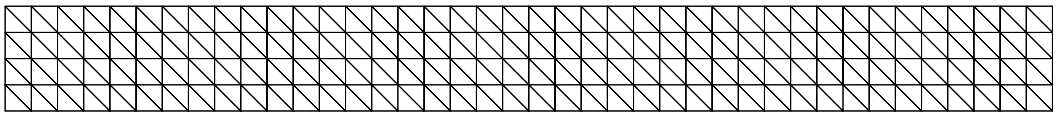


Figure 5.1.3.2: Simple beam mesh

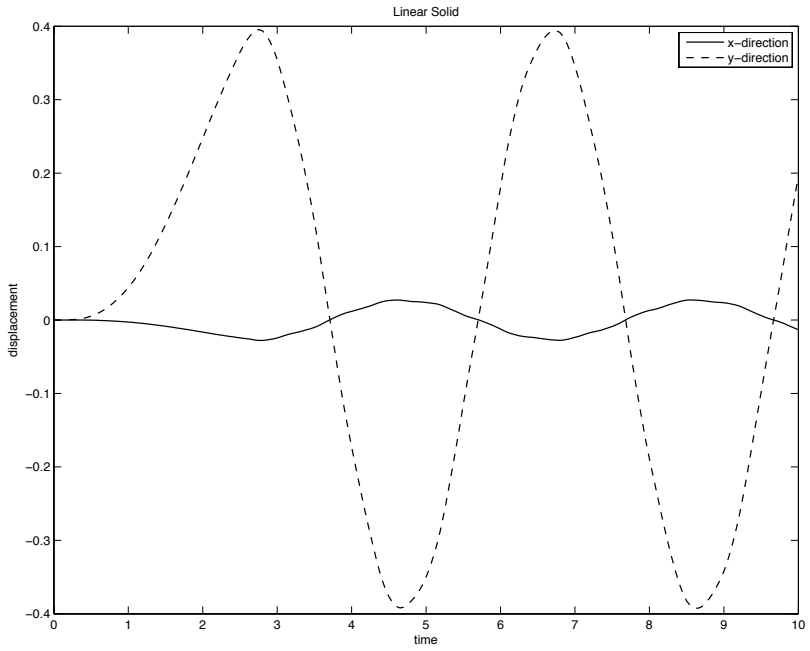


Figure 5.1.3.3: Response of the beam

The input file for SEPMESH is given by:

```
#  beamresponse.msh
#
#  mesh file for time-dependent linear beam response
#  See Manual Standard Elements Section 5.1.3
#  and Examples Manual Section 5.1.3
#  Author: Martijn Booij 2007
#
#  To run this file use:
#     sepmesh beamresponse.msh
#
#  Creates the file meshoutput
#
#  Define some general constants

constants

   integers
      n     = 40      # number of elements in horizontal direction
      m     =  4      # number of elements in vertical direction

      shape_cur = 1   # shape of elements along curves
                      # linear elements
      shape_sur = 3   # shape of elements in surface
                      # linear triangles

   reals
      length = 10     # length of the beam
```

```
      height =  1     # height of the beam
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)              # Left under point
      p2=(length,0)         # Right under point
      p3=(length,height)    # Right upper point
      p4=(0,height)         # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1=line shape_cur (p1,p2,nelm=n)     # lower boundary
      c2=line shape_cur (p2,p3,nelm=m)     # right-hand side boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper
      c4=line shape_cur (p4,p1,nelm=m)     # left-hand side boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4

      s1=rectangle shape_sur (c1,c2,c3,c4)

   plot                          # make a plot of the mesh
                                 # See Users Manual Section 2.2
end
```

Since the load is a function of time we need to supply a function subroutine FUNCCF to define the function. The following program defines this function.

```
      program beamresponse

!     --- Main program for time-dependent linear beam response
!         The main program is standard and consists of 1 statement only
!         See Examples Manual, Section 5.1.3

      call sepcom ( 0 )

      end

!     --- Function subroutine funccf to define the time-dependent load
!         See SEPRAN introduction 5.5.3

      double precision function funccf ( ichoice, x, y, z )
      implicit none

!     --- declaration of input parameters

      integer ichoice
```

```
      double precision x, y, z

!     --- include common ctimen, which contains the time t

      include 'SPcommon/ctimen'

      if ( ichoice==1 ) then

!     --- ichoice = 1, define load

          if ( t<=2.500001d0) then

!         --- t <= 2.5, load is equal to 10t

             funccf = 10*t

          else

!         --- t > 0, no load

             funccf = 0

          end if  ! ( t<=2.5)

       else

!     --- ichoice > 1, not defined

          print *, 'wrong value of ichoice ', ichoice, ' in funccf'
          stop

       end if  ! ( ichoice==1 )

       end
```

The input file for the computational part reads:

```
# beamresponse.prb
#
#  problem file for time-dependent linear beam response
#  See Manual Standard Elements Section 5.1.3
#  and Examples Manual Section 5.1.3
#  Author: Martijn Booij 2007
#
#  To run this file use:
#     sepcomp beamresponse.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
```

```
   integers
      num_int = 3          # Numerical integration rule
   reals
      t0 =       0         # Start time
      t1 =      10         # End time
      dt =      0.001      # Timestep
      tstep =   0.05       # step for output
      rho =     50         # density
      E =       1e6        # E modulus
      nu =      0.4        # Poisson ratio
   vector_names
      u                    # Solution: displacement
      v                    # Derivative of solution: velocity
      a                    # Derivative velocity: acceleration
end
#
#  Define the type of problem to be solved
#
problem                    # See Users Manual Section 3.2.2

   types                   # Define types of elements,
                           # See Users Manual Section 3.2.2
      elgrp1 = 250         # Type number for linear elasticity
                           # See Standard problems Section 5.1

   natbouncond             # Define natural boundary conditions (prescribed load)
      bngrp1 = 251         # Type number for prescribed load

   bounelements            # Define where natural boundary conditions
                           # are given
      belm1 = curves(c3)   # boundary elements along top boundary

   essbouncond             # Define where essential boundary conditions are
                           # given (not the value)
                           # See Users Manual Section 3.2.2
      curves(c4)           # Clamped along left-hand side boundary
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   symmetric               # symmetrical profile matrix
                           # hence a direct solver is used
end

#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 5.1

coefficients

# internal elements

   elgrp1 (nparm = 45)     # The coefficients are defined by 45 parameters
```

```
      icoef2 = 0           # 2d plain strain
      icoef3 =  num_int    # type numerical integration
      coef6 =   E          # E modulus
      coef7 =   nu         # Poisson ratio
      coef43 =  rho        # Density

# boundary elements
   bngrp1 (nparm=25)
      icoef1 = 2           # ILOAD
      icoef2 = 0           # IGPROB
      icoef3 =  num_int    # type numerical integration
      coef6 =  0           # Load in x-direction (0)
      coef7 =  (func= 1)   # Load in y-direction is time dependent
                           # Is defined by funccf with ichoice = 1

end
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary since we need to define initial velocity and displacement

structure

   create_vector  u        # Set initial displacement to 0
   create_vector  v        # Set initial velocity to 0

  #  Solve the time-dependent equations

   solve_time_dependent_problem, vector =  u

end

#  Definition of the time integration scheme
#  See Users Manual Section 3.2.15

time_integration

   method = newmark       # The second order time-derivative is integrated
                          # by the Newmark scheme

   tinit =  t0            # Initial time
   tend =   t1            # End time
   tstep =  dt            # Time step

   toutinit =  t0         # Initial time for output to sepcomp.out
   toutend  =  t1         # End time for output to sepcomp.out
   toutstep =  tstep      # Time step for output to sepcomp.out

   print_level = 2        # Produce some extra output during integration

   beta =  0.25           # Parameter beta for Newmark scheme (default value)
   gamma = 0.5            # Parameter gamma for Newmark scheme (default value)

end

# input for linear solver
```

```
# See Users Manual Section 3.2.8

solve
   positive_definite       # the matrices are positive definite
end

end_of_sepran_input
```

Finally the following input file for seppost may be used.

```
# beamresponse.pst
#
#  postprocessing file for time-dependent linear beam response
#  See Manual Standard Elements Section 5.1.3
#  and Examples Manual Section 5.1.3
#  Author: Martijn Booij 2007
#
#  To run this file use:
#     seppost beamresponse.pst
#
#  Reads the files meshoutput and sepcomp.out

postprocessing
  # Plot time history of both displacements in right-upper point

   time history (0,10), plot point (10,1),  u, degfd 1
   time history (0,10), plot point (10,1),  u, degfd 2

  # Print time history of both displacements in right-upper point

   time history (0,10), print point (10,1), u, degfd 1
   time history (0,10), print point (10,1), u, degfd 2
end
```

5.2 Linear incompressible or nearly incompressible elastic problems

This Chapter is under preparation.

5.3 Nonlinear solid computation

Non-linear solid mechanics problems can be solved either by a Total Lagrange approach or an updated Lagrange approach. In SEPRAN elements for both types of equations are available.
Section (5.3.1) treats elements using the Total Lagrange approach.
Elements using the updated Lagrange approach are treated in Section (5.3.2).

5.3.1 Nonlinear solid computation using a Total Lagrange approach

In this section we treat examples of the total Lagrange approach.
At this moment the following examples are available:

leafspring (5.3.1.1) Computes the displacement of a leafspring.

5.3.1.1 The leafspring example

Consider a leafspring as pictured in Figure 5.3.1.1.1. Point 1 is fixed, point 3 can only slide in x direction. The loading is applied at point 2. The loading can be a prescribed displacement (essential boundary condition) or a prescribed force (natural boundary condition). The front of the spring will be fixed (no translation in y direction), while the back can move freely. For the material of the spring we choose steel ($E = 10MPa, \nu = 0.3$), and the material will behave Hookean for small strains. This material behavior must be programmed in routine FNMATERI. Although the material is isotropic, this example is extended with a local direction in order to demonstrate the usage of FNLOCDIR. In this case it is used to compute strains and stresses in local directions. (For computation of the actual displacements, these local directions are irrelevant).
In this particular example it is not necessary to program the subroutine FNMATERI yourself since the standard (default) subroutine already provides the possibility of using Hookean material. In fact if you leave FNMATERI you will notice no difference.

To get this example into your local directory use:

```
sepgetex leafspring
```

and to run it use:

```
sepmesh leafspring.msh
seplink leafspring
leafspring leafspring.prb
```

In this example you will see that the displacements are large, but the strains are small. So, a linear material model is allowed.

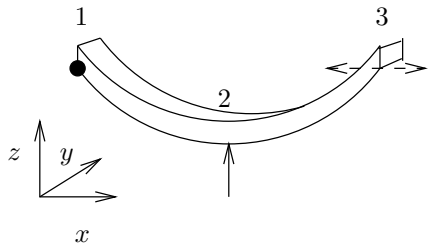


Figure 5.3.1.1.1: leafspring problem

The mesh for the leafspring example may be generated by program sepmesh. Sepmesh requires an input file, for example the file leafspring.msh (5.3.1.1)

Figure 5.3.1.1.2 shows the mesh generated by program sepmesh.

To compute the displacements, program leafspring may be used. This program consists of a simple call to subroutine sepcom only. The reason that this program is used is that the material subroutines FNMATERI and FNLOCDIR must be provided. Otherwise it would be sufficient to call program sepcomp. (5.3.1.1)

In this example we show two different input files, one with respect to prescribed displacements (`leafspring.prb`) and one with respect to prescribed forces (`leafspring1.prb`). In both cases the computed displacements are of the order 0.1 which is relatively big compared to the strains, which are less than 0.01. (5.3.1.1)

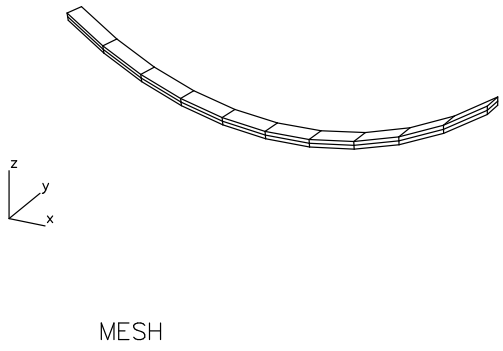


Figure 5.3.1.1.2: Mesh as generated with the mesh input

Mesh file

```
# leafspring.msh
#
#  mesh input file for leafspring example
#
# example for nonlinear solid mechanics.
#  - large displacements
#  - small strains (linear material model)
#  - Total Lagrange approach
#
#
#  See Examples Manual Section 5.3.1.1
#
#  To create the mesh run:
#
#  sepmesh leafspring.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#

constants           # See Users Manual Section 1.4

   integers
      nelx = 5         # number of elements in length of spring
      nelx2= 2*nelx    # 2 nelx
      nely = 2         # number of elements in y-direction
      nelz = 2         # number of elements in z-direction
   reals
      lof  = 0.6       # length of leafspring
      rct  = 0.8       # radius center
      rds  = 1.0       # leafspring radius
      axp  = 0.2       # axis position
      wd   = 0.05      # leaf width
      wh   = 0.02      # leaf heigth
      axmh = axp-wh    # axp - wh
end
#
```

```
#  Define the mesh
#
mesh3d            # See Users Manual Section 2.2
#
#  user points
#
   points             # See Users Manual Section 2.2

      p1 = (   0,   0,  rct )

      p2 = ( -lof,   0, 0    )
      p3 = (    0,   0, -axp )
      p4 = (  lof,   0, 0.0  )

      p5 = ( -lof,   0,  wh  )
      p6 = (    0,   0,-axmh )
      p7 = (  lof,   0,  wh  )

      p8 = ( -lof,  wd, 0    )
      p9 = (    0,  wd, -axp )
      p10 =(  lof,  wd, 0    )

      p11 = ( -lof,  wd,  wh )
      p12 = (   0,  wd,-axmh )
      p13 = (  lof,  wd,  wh )
#
#   curves
#
   curves          # See Users Manual Section 2.3
      c1 = arc1( p2, p3, p1, nelm= nelx )
      c2 = arc1( p3, p4, p1, nelm= nelx )
      c3 = curves ( c1, c2 )
      c4 = translate c3 ( p5, p6, p7 )
      c5 = line1 ( p5, p2, nelm= nely )
      c6 = line1 ( p4, p7, nelm= nely )

      c7 = translate c3 ( p8, p9, p10 )
      c8 = translate c3 ( p11, p12, p13 )
      c9 = line1 (p11, p8, nelm= nely )
      c10= line1 (p10, p13, nelm= nely )

      c11= line1 (p2, p8, nelm= nelz )
      c12= line1 (p4, p10, nelm= nelz )
      c13= line1 (p5, p11, nelm= nelz )
      c14= line1 (p7, p13, nelm= nelz )

      c15= line1 (p3, p6, nelm= nely)
#
#  surface
#
   surfaces       # See Users Manual Section 2.4
      s1 = rectangle5( c3, c6, -c4, c5 )
      s2 = rectangle5( c7, c10,-c8, c9 )
      s3 = coons5(c3, c12,-c7,-c11 )
      s4 = coons5(c4, c14,-c8,-c13)
```

```
      s5 = rectangle5( c11,-c9,-c13,c5)
      s6 = rectangle5( c12,c10,-c14,-c6)
#
#  volume
#
   volumes       # See Users Manual Section 2.5
      v1 = brick13( s3, s1, s6, s2, s5, s4 )


   plot           # make a plot of the mesh
                  # See Users Manual Section 2.2

end
```

Program

```
      program leafspring
!     program file for leafspring example
!     See Examples Manual Section 5.3.1.1
      call sepcom( 0 )
      end

      subroutine fnmateri ( ichoice, s, se, eps, detf, matpar, makese )
! ======================================================================
!
!
!
!                       DESCRIPTION
!
!     MATERI :  routine for ELM250. Material behaviour in nonlinear case
!
!     Compute 2nd-Piola-Kirchoff stress from given Green-Lagrange
!     strains. The determinant of the deformation gradient, can be
!     used in this relationship
!
!       EXAMPLE MATERIAL
!       linear elastic material model (isotropic hooke)
!       the large displacement/rotation are taken into account
!       in the Green-Lagrange strain, which is small.
!
! **********************************************************************
!
!                       KEYWORDS
!
!       elasticity
!       nonlinear
! **********************************************************************
!
!                       INPUT / OUTPUT PARAMETERS
!
      logical          makese
      double precision s(6), eps(6), detf, matpar(10),
     +                 se(6,6)
      integer ichoice

!     detf  i    determinant of deformation gradient
!     eps   i    Green-Lagrange strains
!                   eps(i)  : i: components (symmetric!);
!     ichoice i   material model number ( icoef 4 )
!                   0 : hookean
!                        coef6 = E
!                        coef7 = nu
!     makese i   se must be computed (true) or not (false)
!     matpar i   material parameters (User defines!) for every
!                integration point
!     nip   i    number of data points
!     s     o    2nd-PK-stresses: You only have to store symmetric
!                components!
!                   s(j) :    j: component (1=11, 2=22, 3=33,
```

```
!                                          4=12, 5=23, 6=31)
!     se    o    tangential matrix.
!                   se(i,j) : i,j components (symmetric!)
! ********************************************************************
!
!                       COMMON BLOCKS
!
! ********************************************************************
!
!                       LOCAL PARAMETERS

      integer          i, j
      double precision a0, a1, a2, a3, E, nu

! ********************************************************************
!
!                       SUBROUTINES CALLED
!
! ********************************************************************
!
!                       I/O
!
! ********************************************************************
!
!                       ERROR MESSAGES
!
! ********************************************************************
!
!                       PSEUDO CODE
!
! ********************************************************************
!
!                       DATA STATEMENTS
!
! ====================================================================
!
!     --- material parameters

      E  = matpar(1)
      nu = matpar(2)
      a0 = E / ( 1d0 + nu )
      a1 = a0 * ( 1d0 - nu ) /( 1d0 - 2d0*nu)
      a2 = a0 * nu /( 1d0 - 2d0*nu)
      a3 = a0 / 2d0

!     --- clear se and s

      do i = 1, 6
         s(i) = 0d0
         do j = 1, 6
            se(i,j) = 0d0
         end do
      end do

      do i=1,3
```

```
         se(i,i) = a1
         se(i+3,i+3) = a3
      end do

      se(1,2) = a2
      se(2,1) = a2
      se(2,3) = a2
      se(3,2) = a2
      se(1,3) = a2
      se(3,1) = a2

      do i=1,3
         do j=1,3
            s(i) = s(i) + se(i,j)*eps(j)
         end do
         s(i+3) = se(i+3,i+3)*eps(i+3)
      end do

      do i = 1, 3
         se(i,i) = a1
         se(i+3,i+3) = a3*0.5d0
      end do

      end

      subroutine fnlocdir ( pos, dir, ielgrp )

!     --- example for usage of fnlocdir

      implicit none
      double precision pos(3), dir(3,3)
      integer ielgrp

      double precision rct, x, y, z, len

      rct = 0.8d0
      x = pos(1)
      y = pos(2)
      z = pos(3) + rct
      len = sqrt( z*z + x*x )

!     --- local x-vector

      dir(1,1) = z/len
      dir(2,1) = 0d0
      dir(3,1) = -x/len

!     --- local y-vector (unchanged)

      dir(1,2) = 0d0
      dir(2,2) = 1d0
      dir(3,2) = 0d0

!     --- local z-vector
```

```
dir(1,3) = -dir(3,1)
dir(2,3) = 0d0
dir(3,3) = dir(1,1)

end
```

Problem definition file

```
# leafspring.prb
#
#  problem input file for leafspring example
#  See Examples Manual Section 5.3.1.1
#
# example for nonlinear solid mechanics.
#  - large displacements
#  - small strains (linear material model)
#  - Total Lagrange approach
#
#  To run this file use:
#     seplink leafspring
#     leafspring leafspring.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#  Define some general constants
#

constants           # See Users Manual Section 1.4
   vector_names
      incr_displacement   # the incremental displacement in each step
      tot_displacement    # contains the total displacement
      strain              # contains the strain
      stress              # contains the stress
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2
                          # solves the velocity (momentum equations: predictor)

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2

      elgrp1, (type=250)       # element type for solid material

   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
                               # suppressed displacements and prescribed displ.
      degfd3 = points (p3,p9)
      degfd1 = curves (c11)
      degfd3 = curves (c11)
      degfd3 = curves (c12)
      degfd2 = surfaces (s1)
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4
matrix
```

```
   symmetric     # Symmetrical profile matrix
                 # So a direct solver is applied
end
#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#
structure

#  First initialize both vectors
   create_vector, tot_displacement
   prescribe_boundary_conditions, incr_displacement
#  Solve system of equations
   solve_nonlinear_system, incr_displacement
   print tot_displacement, text='displacement'
#  Compute stresses and strains
   derivatives, seq_deriv=1,  strain
   print strain
   derivatives, seq_deriv=2,  stress
   print stress

#     The vectors are written to the file sepcomp.out
#     The vector incr_displacement is skipped

   output, vector = tot_displacement

end
#
#  essential boundary conditions:
#
essential boundary conditions
   points (p3,p9), degfd3 = (value=0.10)
end

#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 5.3.2

coefficients
   elgrp1 (nparm=45)
     icoef 2 = 10
     coef  6 = ( value = 1d7 )
     coef  7 = ( value = 0.3 )
end
#
# Information concerning the non-linear problem
#
nonlinear_equations
    global_options, maxiter=25, accuracy=1d-4, print_level=2//
                    iteration_method = incremental_newton//
                    seqtotal_vector= tot_displacement
    equation 1
       fill_coefficients = 1
 end
#
```

```
# Information about the derivatives to be computed
#
 derivatives
    icheld=7
    seq_input_vector = tot_displacement
 end
 derivatives, sequence_number=2
    icheld=6
    seq_input_vector = tot_displacement
 end
end_of_sepran_input
```

5.3.2 Nonlinear solid computation using an updated Lagrange approach

In this section we treat examples of the updated Lagrange approach.
At this moment the following examples are available:

beam2d (5.3.2.1) Bending of a beam (2D)

block2d (5.3.2.2) Deformation with volume change of a block (2D)

artery2d (5.3.2.3) Arterial wall with internal pressure (2D)

block3d (5.3.2.4) Uni-axial tension test (3D)

artery3d (5.3.2.5) Arterial wall with internal pressure (3D)

5.3.2.1 Bending of a beam (2D)

In this example the deformation of a 2D solid beam is demonstrated. Both rotations and strains are large. The material behavior is described by a hyper elastic incompressible Neo-Hookean material law as described in the manual Standard Problems Section 5.3.2 (element type 202). The elements chosen for the mesh are 9-noded quadratic quadrilaterals (shape=6). The beam is fixed at the bottom edge by applying homogeneous dirichlet boundary conditions. Along the left edge a normal force is applied. Since this problem is geometrically non-linear the force is increased gradually and each force-increment a new equilibrium is computed using a Newton-Raphson iterative loop. Note that the normal direction of the force also varies depending on the deformation of the beam during each iteration step. The mesh for this problem with the corresponding boundary conditions is shown in figure 5.3.2.1.1 on the left. On the right in this figure the deformation of the beam at increasing force is shown going from A-E.

To get this example into your local directory use:

```
sepgetex beam2d
```

and to run it use:

```
sepmesh beam2d.msh
sepcomp beam2d.prb
```

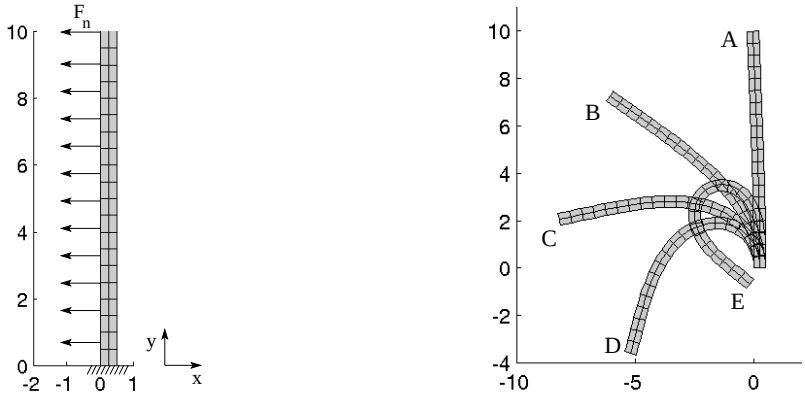


Figure 5.3.2.1.1: Mesh with boundary conditions (left) and corresponding deformations for different values of the normal force (right)

In this problem the displacements are large (geometrically non-linear deformation) and therefore the updated Lagrange approach is used.

The method works as follows:
A pseudo time integration is applied, in which the time is only used to increase the applied pressure. In this example we use 10 time steps and the pressure is increased from 0 to 4, by making it equal to $10t$.
In each pseudo time step, we have to solve a non-linear iteration procedure.
In this method we need the following displacement vectors:

un The total displacement

u The displacement per time step.
This one is used to increment the total displacement.

δu the incremental displacement vector per iteration.
This one is used to increment the displacement per time step.

The method can be explained in the following algorithm

Create the mesh
Set t_0, Δt and $tend$
un := 0
$t := t_0$; end_time_loop := false
while not end_time_loop **do**
  t := $t + \Delta t$ Clear **u**
  while not converged **do**
    Clear δu
    Build matrix and right-hand side based on **u** and **un**
    Solve system of equations to get new δu
    u := **u** + δu
  end while
  un := **un** + **u**
  end_time_loop := $t \leq tend$
end while

The element assumes that **u** and **un** are the first 2 vectors in the list of vectors.

In order to construct the applied pressure as function of the time, we introduce two scalars $incr$ and $force$.
$incr$ is used to count the number of time steps performed and
$force$ is used to store the value of $-0.1\ t$.
In the next pages the mesh file and the problem input file are given

Mesh file

```
#  beam2d.msh
#
#  mesh input file for a 2D beam with a 1:20 ratio
#  geometrically non-linear deformation of a beam.
#  Bending of a beam by applying an external force in normal direction
#
#  The updated Lagrange approach is used
#  This includes a pseudo time loop, with non-linear iteration per step
#  The pressure is increased during the time stepping
#
#  See Manual Standard Elements Section 5.3.2
#  and examples manual Section 5.3.2.1
#
#  To create the mesh run:
#
#  sepmesh beam2d.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#

constants           # See Users Manual Section 1.4

   integers
     nelemx = 2   # number of elements over width of beam
     nelemy = 20  # number of elements over height of beam
     lin=2        # ishape => quadratic line elements
     surf=6       # ishape => quadratic quadrilaterals

   reals
     xx1 =  0    # origin
     xx2 =  .5   # width of beam
     yy1 =  0    # origin
     yy2 =  10   # height of beam

end
#
#  Define the mesh
#
mesh2d            # See Users Manual Section 2.2
#
#  user points
#
   points             # See Users Manual Section 2.2
      p1 = ( xx1, yy1)  # left bottom point
      p2 = ( xx2, yy1)  # right bottom point
      p3 = ( xx2, yy2)  # right upper point
      p4 = ( xx1, yy2)  # left upper point
#
#   curves
#
   curves          # See Users Manual Section 2.3
```

```
      c1  = line lin(p1,p2,nelm= nelemx)  # bottom curve
      c2  = line lin(p2,p3,nelm= nelemy)  # right curve
      c3  = line lin(p3,p4,nelm= nelemx)  # upper curve
      c4  = line lin(p4,p1,nelm= nelemy)  # left curve
#
#  surface
#
   surfaces       # See Users Manual Section 2.4

      s1 = rectangle surf( c1, c2,  c3, c4)

   plot           # make a plot of the mesh
                  # See Users Manual Section 2.2

end
```

Problem definition file

```
#  beam2d.prb
#
#  geometrically non-linear deformation of a 2D plain-strain artery.
#  A pressure is applied at the inner arterial wall.
#
#  The updated Lagrange approach is used
#  This includes a pseudo time loop, with non-linear iteration per step
#  The pressure is increased during the time stepping
#
#  See Manual Standard Elements Section 5.3.2
#  and examples manual Section 5.3.2.1
#
#  To run this file use:
#     sepcomp  beam2d.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#  Suppress superfluous output

set warn off
set output none

#
#  Define some general constants
#

constants           # See Users Manual Section 1.4

   integers
      nincr = 400       # number of increments
   reals
      tstart = 0           # start of artificial time algorithm
      tend = 4             # end of artificial time algorithm
      dt =   tend/ nincr   # artificial time step defined by tend/nincr
   variables
      incr = 0             # counter for increments (used for printing only)
      force = 0            # is used to define pressure at inner wall
   vector_names
    #   The following vectors are used for the computation of the displacement
    #   Mark that the vectors u and un must always be given
    #   as first and second vector
      u          # Displacement vector per pseudo time step
      un         # Total displacement vector
      stress     # stress
      strain     # strain
      pressure   # pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2
```

```
                          # solves the velocity (momentum equations: predictor)

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2

      elgrp1 = 202         # element type for solid material
                           # updated Lagrange approach
                           # Taylor-Hood elements (continuous pressure)

   natbouncond             # Definition of type numbers for natural
                           # boundary conditions
      bngrp1 = 210         # boundary group used to apply internal pressure

   bounelements            # Definition of boundary elements
      belm1 = curves(c4)     # curve at which force is applied

   essbouncond                # Define where essential boundary conditions are
                              # given (not the value)
                              # See Users Manual Section 3.2.2
      degfd1,degfd2 = curves(c1)   # fix bottom curve of beam

    renumber levels (1,2),(3)  # renumbering of unknowns per level
                               # first displacements, then pressures
                               # in this way zero pivots are avoided
end

#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 5.3.2

coefficients
  # internal elements
   elgrp1 (nparm = 45)      # The coefficients are defined by 45 parameters
      icoef2 = 0            # type of stress-strain relation
                            # 0 - 2D plane strain
      icoef3 = 0            # type of numerical integration
                            # 0 - default value
      icoef4 = 2            # constitutive law
                            # 2  - incompressible Neo-Hookean
      icoef5 = 0            # user flags
                            # iusrvc = 0 - user vector is not filled
      coef7 = 1             # Take into account the linearization of
                            # the Jacobian
      coef10 = 1d4
                            # Mooney-Rivlin: material parameter c0


  # boundary elements
   bngrp1 (nparm = 25)      # The coefficients are defined by 25 parameters
      icoef1= 2             # 2 - local coor system with linearization
                            #     for boundary conditions
      icoef3=1              # Integration rule (1=Newton Cotes)
      coef6=  force         # force in normal direction as a function (p=0.1 t)

end
```

```
#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#
structure

   ### Create total displacement vector   and set to 0
   create_vector,  un

   ### Start incremental (pseudo-time) loop
   start_time_loop

      ### Clear solution vector (Displacement vector per pseudo time step)
      create_vector,  u

      time_integration  # Adjust the time parameters
                                          # No actual action

      incr = incr+1                       # Raise increment counter
                                          # t = incr*dt
      force = -10*incr*dt                 # Compute force as function of t

      ### Print time and increment number
      print_time
      print incr, text = 'increment'

      ### Solve system of non-linear equations to get new increment vector
      solve_nonlinear_system,  u

      ### Compute stress, strain and pressure vectors
      #   This must be done before updating the mesh and un

      derivatives, seq_deriv = 1,  stress
      derivatives, seq_deriv = 2,  strain
      derivatives, seq_deriv = 3,  pressure

      ### Deform mesh using the displacement vector
      deform_mesh,  u

      ### Update total solution vector
      un =  un +  u

   ### End time loop
   end_time_loop

end

#  Definition of (pseudo) time integration
#  See Users Manual Section 3.2.15

time_integration
   method = stationary    # no action, just adjusting time parameters
   tinit =  tstart        # start time
   tend =  tend           # end time
   tstep =  dt            # time step
```

```
end


# Definition of iteration for non linear equations
#  See Users Manual Section 3.2.9

nonlinear_equations
   global_options, maxiter = 50, miniter = 1, accuracy = 1d-3//
      criterion = relative,  print_level = 2,  at_error= return //
      iteration_method = newton
   equations 1
     fill_coefficients = 1
end

#  Compute stress tensor
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives
  icheld = 6                    # compute stress
  seq_input_vector =  u         # use the total displacement as input vector
end

#  Compute stress tensor
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives, sequence_number = 2
  icheld = 8                    # compute strain
  seq_input_vector =  u         # use the total displacement as input vector
end

#  Compute pressure
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives, sequence_number = 3
  icheld = 7                    # compute pressure
  seq_input_vector =  u         # use the total displacement as input vector
end
```

5.3.2.2 Deformation with volume change of a block (2D)

In this example the deformation and volumetric change of a 2D block are presented. A hyper-elastic compressible Neo-Hookean material model is used to describe the material behavior as described in the manual Standard Problems Section 5.3.2 (element type 200). The mesh for this block consists of 7-noded quadratic triangles (shape=7)(figure 5.3.2.2.1 on the left). The bottom and the left side of the block are fixed in y and x-direction, respectively. At the top and right side the force in y and x-direction is applied, respectively. This force is enforced incrementally and is a function of the coordinates. The resulting deformations of the block with the red lines denoting the original shape, are shown in figure 5.3.2.2.1 on the right.

To get this example into your local directory use:

```
sepgetex block2d
```

and to run it use:

```
sepmesh block2d.msh
seplink block2d
block2d < block2d.prb
```

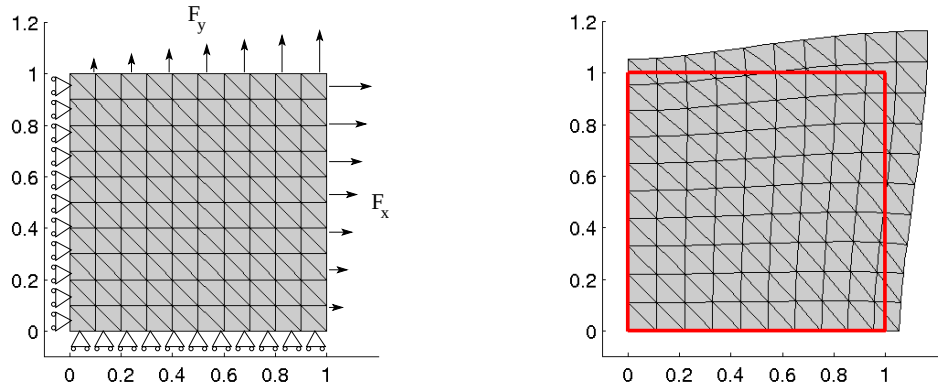


Figure 5.3.2.2.1: Mesh with boundary conditions (left) and corresponding deformations with red line denoting the original form (right)

In this problem the displacements are large (geometrically non-linear deformation) and therefore the updated Lagrange approach is used.

Exactly the same method as in Section 5.3.2.1 is used.
The pressure at the boundary depends on space and so we need a function subroutine FUNCCF and consequently a main program.

The files are given at the following pages

Mesh file

```
#  block2d.msh
#
#  mesh input file for a 2D block with a 1:1 ratio
#  geometrically non-linear deformation of a (2D plain-strain) block.
#  The compressible material is stretched from two sides resulting in
#  a volume increase
#
#  The updated Lagrange approach is used
#  This includes a pseudo time loop, with non-linear iteration per step
#  The pressure is increased during the time stepping
#
#  See Manual Standard Elements Section 5.3.2.2
#  and examples manual Section 5.3.2.2
#
#  To create the mesh run:
#
#  sepmesh block2d.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4

   integers

     nelemx = 10   # number of elements in x-direction
     nelemy = 10   # number of elements in y-direction
     lin=2         # ishape => quadratic line element
     surf=7        # ishape => extended quadratic triangular element

   reals
     xx1 =  0
     xx2 =  1   # width of block
     yy1 =  0
     yy2 =  1   # height of block
end
#
#  Define the mesh
#
mesh2d            # See Users Manual Section 2.2
#
#  user points
#
   points             # See Users Manual Section 2.2

      p1 = ( xx1, yy1)   # left bottom point
      p2 = ( xx2, yy1)   # right bottom point
      p3 = ( xx2, yy2)   # right upper point
      p4 = ( xx1, yy2)   # left upper point
#
#   curves
#
```

```
   curves          # See Users Manual Section 2.3
      c1  = line lin(p1,p2,nelm= nelemx)   # bottom curve
      c2  = line lin(p2,p3,nelm= nelemy)   # right curve
      c3  = line lin(p3,p4,nelm= nelemx)   # upper curve
      c4  = line lin(p4,p1,nelm= nelemy)   # left curve
#
#  surface
#
   surfaces       # See Users Manual Section 2.4

      s1 = rectangle surf( c1, c2,  c3, c4)    # total surface

   plot           # make a plot of the mesh
                  # See Users Manual Section 2.2

end
```

Program

```
      program block2d
!     main program for
!     geometrically non-linear deformation of a (2D plain-strain) block.
!     The compressible material is stretched from two sides resulting in
!     a volume increase
!
!     The updated Lagrange approach is used
!     This includes a pseudo time loop, with non-linear iteration per step
!     The pressure is increased during the time stepping
!
!     See Manual Standard Elements Section 5.3.2
!     and examples manual Section 5.3.2.2
      call sepcom ( 0 )
      end

      double precision function funccf ( ichoice, x, y, z )
      implicit none

      include 'SPcommon/ctimen'

      integer ichoice;
      double precision x, y, z, c1, c2

      c1= 2d1
      c2= 2d1

      if ( ichoice==1 ) then
         funccf = c1*y*t
      else if ( ichoice==2 ) then
         funccf = c2*x*t
      end if

      end
```

Problem definition file

```
#  block2d.prb
#
#  geometrically non-linear deformation of a (2D plain-strain) block.
#  The compressible material is stretched from two sides resulting in
#  a volume increase
#
#  The updated Lagrange approach is used
#  This includes a pseudo time loop, with non-linear iteration per step
#  The pressure is increased during the time stepping
#
#  See Manual Standard Elements Section 5.3.2
#  and examples manual Section 5.3.2.2
#
#  To run this file use:
#     seplink block2d
#     block2d < block2d.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#  Suppress superfluous output

set warn off
set output none

#
#  Define some general constants
#

constants           # See Users Manual Section 1.4

   integers
      nincr = 20       # number of increments
   reals
      tstart = 0           # start of artificial time algorithm
      tend = 1             # end of artificial time algorithm
      dt =   tend/ nincr   # artificial time step defined by tend/nincr
   variables
      incr = 0             # counter for increments (used for printing only)
   vector_names
    #   The following vectors are used for the computation of the displacement
    #   Mark that the vectors u and un must always be given
    #   as first and second vector
      u          # Displacement vector per pseudo time step
      un         # Total displacement vector
      stress     # stress
      strain     # strain
      pressure   # pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2
```

```
                          # solves the velocity (momentum equations: predictor)

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2

      elgrp1 = 200         # element type for solid material
                           # updated Lagrange approach
                           # Compressible material
   natbouncond             # Definition of type numbers for natural
                           # boundary conditions

    bngrp1 = 210           # boundary group to apply force
    bngrp2 = 210           # boundary group to apply force
   bounelements            # Definition of boundary elements
     belm1 = curves(c2)              # apply force on right side
     belm2 = curves(c3)              # apply force on upper side
   essbouncond                # Define where essential boundary conditions are
                              # given (not the value)
                              # See Users Manual Section 3.2.2
      degfd1 = curves(c4)   # fix left side in x-direction
      degfd2 = curves(c1)   # fix bottom side in y-direction
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   storage_method = compact   # Non-symmetrical compact matrix
               # So an iterative solver is applied
end

#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 5.3.2

coefficients
  # internal elements
   elgrp1 (nparm = 45)      # The coefficients are defined by 45 parameters
      icoef2 = 0            # type of stress-strain relation
                            # 0 - 2D plane strain
      icoef3 = 0            # type of numerical integration
                            # 0 - default value
      icoef4 = 1            # constitutive law
                            # 1  - compressible Neo-Hookean
      icoef5 = 0            # user flags
                            # iusrvc = 0 - user vector is not filled
      coef7 = 1             # Take into account the linearization of
                            # the Jacobian
      coef10 = 40           # shear modulus
      coef11 = 40           # bulk modulus

   # coefficients concerning the first boundary group
   bngrp1 (nparm = 25)
      icoef1= 0       # 0 - global coordinate system for boundary conditions
      icoef3=1        # Integration rule (1=Newton Cotes)
```

```
      coef6= func=1   # force in global x-direction as a function (see block2d.f)
      coef7= 0d0      # force in global y-direction

   # coefficients concerning the second boundary group
   bngrp2 (nparm = 25)
      icoef1= 0       # 0 - global coordinate system for boundary conditions
      icoef3=1        # Integration rule (1=Newton Cotes)
      coef6= 0d0      # force in global x-direction
      coef7= func=2   # force in global y-direction as a function (see block2d.f)

end

structure

   ### Create total displacement vector   and set to 0
   create_vector,  un

   ### Start incremental (pseudo-time) loop
   start_time_loop

      ### Clear solution vector (Displacement vector per pseudo time step)
      create_vector,  u

      time_integration  # Adjust the time parameters
                                          # No actual action

      incr = incr+1                       # Raise increment counter
                                          # t = incr*dt

      ### Print time and increment number
      print_time
      print incr, text = 'increment'

      ### Solve system of non-linear equations to get new increment vector
      solve_nonlinear_system,  u

      ### Compute stress, strain and pressure vectors
      #   This must be done before updating the mesh and un

      derivatives, seq_deriv = 1,  stress
      derivatives, seq_deriv = 2,  strain
      derivatives, seq_deriv = 3,  pressure

      ### Deform mesh using the displacement vector
      deform_mesh,  u

      ### Update total solution vector
      un =  un +  u

   ### End time loop
   end_time_loop

end

#  Definition of (pseudo) time integration
```

```
#  See Users Manual Section 3.2.15

time_integration
   method = stationary    # no action, just adjusting time parameters
   tinit =  tstart        # start time
   tend =  tend           # end time
   tstep =  dt            # time step
end

# Definition of iteration for non linear equations
#  See Users Manual Section 3.2.9

nonlinear_equations
   global_options, maxiter = 50, miniter = 1, accuracy = 1d-3//
      criterion = relative,  print_level = 2,  at_error= return //
      iteration_method = newton
   equations 1
     fill_coefficients = 1
end

#  Compute stress tensor
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives
  icheld = 6                    # compute stress
  seq_input_vector =  u         # use the total displacement as input vector
end

#  Compute stress tensor
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives, sequence_number = 2
  icheld = 8                    # compute strain
  seq_input_vector =  u         # use the total displacement as input vector
end

#  Compute pressure
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives, sequence_number = 3
  icheld = 7                    # compute pressure
  seq_input_vector =  u         # use the total displacement as input vector
end

#  Information for linear solver
#  See Users Manual Section 3.2.8

solve
    iterative_method = BICGSTAB, preconditioner = ilu
end
```

5.3.2.3 Arterial wall with internal pressure (2D)

In this problem an 2D arterial wall is considered that deforms by an internal pressure. A hyper-elastic incompressible Neo-Hookean material law describes the material behavior (element type 201). The mesh is subdivided into 9-noded quadratic quadrilaterals (shape=6). The boundary conditions corresponding to this problem are shown together with the mesh in Figure 5.3.2.3.1 on the left. An internal pressure is applied at the inner wall and the displacements at some of the radial curves are prescribed to fix the mesh in space. On the right side of Figure 5.3.2.3.1 the deformation of the mesh is shown with the corresponding pressure contour bands.

To get this example into your local directory use:

```
sepgetex artery2d
```

and to run it use:

```
sepmesh artery2d.msh
sepcomp artery2d.prb
```

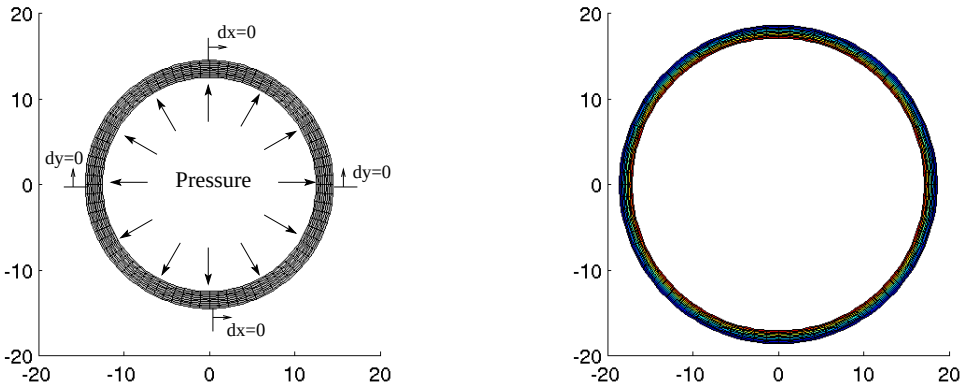


Figure 5.3.2.3.1: Mesh with boundary conditions (left) and deformations with pressure contour-bands (right)

The mesh is created by splitting the region into four parts. This is not necessary but makes the prescription of boundary conditions more easy.
The method used to solve the problem is exactly identical to the one in Section 5.3.2.1. The only difference is that the force is equal to -0.1t and that the time increases form 0 to 1, with steps 0.1.

On the next pages the input files can be found.

Mesh file

```
#  artery2d.msh
#
#  mesh input file for
#  geometrically non-linear deformation of a 2D plain-strain artery.
#  A pressure is applied at the inner arterial wall.
#
#  The region is enclosed by two concentric circles
#
#  The updated Lagrange approach is used
#  This includes a pseudo time loop, with non-linear iteration per step
#  The pressure is increased during the time stepping
#
#  See Examples Manual Section 5.3.2.3
#
#  To create the mesh run:
#
#  sepmesh artery2d.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#

constants           # See Users Manual Section 1.4

  integers
     nelmarc = 20       # number of elements in 1/4 circumferential
     nelmw   = 10       # number of elements in artery wall thickness
     line = 2           # quadratic line elements
     surf = 6           # quadratic quadrilaterals
     inner_circ = 20    # sequence number of inner circle
     outer_circ = 21    # sequence number of outer circle

  reals
     inner_radius = 12.5   # Radius of inner circle
     outer_radius = 14.5   # Radius of outer circle
end
#
#  Define the mesh
#
mesh2d            # See Users Manual Section 2.2
#
#  user points
#
   points             # See Users Manual Section 2.2
   # Centre
     p1 = (0,0)
   # Inner circle: subdivided into 4 parts because of boundary conditions
     p2 = ( inner_radius,0)
     p3 = (0, inner_radius)
     p4 = (- inner_radius,0)
     p5 = (0,- inner_radius)
   # Outer circle: subdivided into 4 parts because of boundary conditions
```

```
     p6 = ( outer_radius,0)
     p7 = (0, outer_radius)
     p8 = (- outer_radius,0)
     p9 = (0,- outer_radius)

#
#   curves
#
   curves          # See Users Manual Section 2.3
    # inner circle, consists of 4 arcs
      c1 = arc  line (p2,p3,p1,nelm =  nelmarc)
      c2 = arc  line (p3,p4,p1,nelm =  nelmarc)
      c3 = arc  line (p4,p5,p1,nelm =  nelmarc)
      c4 = arc  line (p5,p2,p1,nelm =  nelmarc)
      c inner_circ = curves (c1,c2,c3,c4)
    # outer circle, consists of 4 arcs
      c5 = arc line(p6,p7,p1,nelm =  nelmarc)
      c6 = arc line(p7,p8,p1,nelm =  nelmarc)
      c7 = arc line(p8,p9,p1,nelm =  nelmarc)
      c8 = arc line(p9,p6,p1,nelm =  nelmarc)
      c outer_circ = curves (c5,c6,c7,c8)
    # connection, used for boundary conditions
      c9 = line line(p2,p6, nelm =  nelmw)
      c10 = line line(p3,p7,nelm =  nelmw)
      c11 = line line(p4,p8,nelm =  nelmw)
      c12 = line line(p5,p9,nelm =  nelmw)
#
#  surface
#
   surfaces       # See Users Manual Section 2.4
                  # Created from 4 parts

     s1 = quadrilateral surf(c9,c5,-c10,-c1)
     s2 = quadrilateral surf(c10,c6,-c11,-c2)
     s3 = quadrilateral surf(c11,c7,-c12,-c3)
     s4 = quadrilateral surf(c12,c8,-c9,-c4)

   plot           # make a plot of the mesh
                  # See Users Manual Section 2.2

end
```

Problem definition file

```
#  artery2d.prb
#
#  geometrically non-linear deformation of a 2D plain-strain artery.
#  A pressure is applied at the inner arterial wall.
#
#  The updated Lagrange approach is used
#  This includes a pseudo time loop, with non-linear iteration per step
#  The pressure is increased during the time stepping
#
#  See Manual Standard Elements Section 5.3.2
#  and examples manual Section 5.3.2.3
#
#  To run this file use:
#     sepcomp artery2d.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#  Suppress superfluous output

set warn off
set output none

#
#  Define some general constants
#

constants           # See Users Manual Section 1.4
   integers
      nincr = 10           # number of artificial time steps
      inner_circ = 20      # sequence number of inner circle
   reals
      tstart = 0           # start of artificial time algorithm
      tend = 1             # end of artificial time algorithm
      dt =   tend/ nincr   # artificial time step defined by tend/nincr
   variables
      incr = 0             # counter for increments (used for printing only)
      force = 0            # is used to define pressure at inner wall
   vector_names
    #   The following vectors are used for the computation of the displacement
    #   Mark that the vectors u and un must always be given
    #   as first and second vector
      u          # Displacement vector per pseudo time step
      un         # Total displacement vector
    #   Output vectors
      stress     # stress
      strain     # strain
      pressure   # pressure
end
#
#  Define the type of problem to be solved
#
```

```
problem                    # See Users Manual Section 3.2.2
                           # solves the velocity (momentum equations: predictor)

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2

      elgrp1 = 201         # element type for solid material
                           # updated Lagrange approach
                           # Crouzeix-Raviart elements (discontinuous pressure)

   natbouncond             # Definition of type numbers for natural
                           # boundary conditions
      bngrp1 = 210         # boundary group used to apply internal pressure

   bounelements            # Definition of boundary elements
      belm1 = curves(c inner_circ)    # apply pressure on inner boundary

   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      degfd2 = curves(c9)      # fix right radial curve in y-direction
      degfd2 = curves(c11)     # fix left radial curve in y-direction
      degfd1 = curves(c10)     # fix upper radial curve in x-direction
      degfd1 = curves(c12)     # fix bottom radial curve in x-direction
   renumber levels (1,2),(3,4,5)  # renumbering of unknowns per level
                               # first displacements, then pressures
                               # in this way zero pivots are avoided
end

#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 5.3.2

coefficients
  # internal elements
   elgrp1 (nparm = 45)      # The coefficients are defined by 45 parameters
      icoef2 = 0            # type of stress-strain relation
                            # 0 - 2D plane strain
      icoef3 = 0            # type of numerical integration
                            # 0 - default value
      icoef4 = 2            # constitutive law
                            # 2  - incompressible Neo-Hookean
      icoef5 = 0            # user flags
                            # iusrvc = 0 - user vector is not filled
      coef7 = 1             # Take into account the linearization of
                            # the Jacobian
      coef10 = 1            # shear modulus


  # boundary elements
   bngrp1 (nparm = 25)      # The coefficients are defined by 25 parameters
      icoef1= 2             # 2 - local coor system with linearization
                            #     for boundary conditions
      icoef3=1              # Integration rule (1=Newton Cotes)
      coef6=  force         # force in normal direction as a function (p=0.1 t)
```

```
end
#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#
structure

   ### Create total displacement vector   and set to 0
   create_vector,  un

   ### Start incremental (pseudo-time) loop
   start_time_loop

      ### Clear solution vector (Displacement vector per pseudo time step)
      create_vector,  u

      time_integration  # Adjust the time parameters
                                          # No actual action

      incr = incr+1                       # Raise increment counter
                                          # t = incr*dt
      force = -0.1*incr*dt                # Compute force as function of t

      ### Print time and increment number
      print_time
      print incr, text = 'increment'

      ### Solve system of non-linear equations to get new increment vector
      solve_nonlinear_system,  u

      ### Compute stress, strain and pressure vectors
      #   This must be done before updating the mesh and un

      derivatives, seq_deriv = 1,  stress
      derivatives, seq_deriv = 2,  strain
      derivatives, seq_deriv = 3,  pressure

      ### Deform mesh using the displacement vector
      deform_mesh,  u

      ### Update total solution vector
      un =  un +  u

     ### End time loop

   end_time_loop

end

#  Definition of (pseudo) time integration
#  See Users Manual Section 3.2.15

time_integration
   method = stationary    # no action, just adjusting time parameters
```

```
   tinit =  tstart        # start time
   tend =  tend           # end time
   tstep =  dt            # time step
end


# Definition of iteration for non linear equations
#  See Users Manual Section 3.2.9

nonlinear_equations
   global_options, maxiter = 50, miniter = 1, accuracy = 1d-3//
      criterion = relative,  print_level = 2,  at_error= return //
      iteration_method = newton
   equations 1
     fill_coefficients = 1
end

#  Compute stress tensor
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives
  icheld = 6                    # compute stress
  seq_input_vector =  u         # use the total displacement as input vector
end

#  Compute stress tensor
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives, sequence_number = 2
  icheld = 8                    # compute strain
  seq_input_vector =  u         # use the total displacement as input vector
end

#  Compute pressure
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives, sequence_number = 3
  icheld = 7                    # compute pressure
  seq_input_vector =  u         # use the total displacement as input vector
end
```

5.3.2.4 Uni-axial tension test (3D)

In this example a uni-axial tension test is simulated for a 3D solid block. The material is described by an incompressible Neo-Hookean material law as described in the manual Standard Problems Section 5.3.2 (element type 201).

To get this example into your local directory use:

```
sepgetex block3d
```

and to run it use:

```
sepmesh block3d.msh
seplink block3d
block3d < block3d.prb
```

27-noded quadratic hexahedrons (shape=14) are used for the mesh. The mesh is fixed at the bottom, left and front surface (these surfaces fall within the x-y, y-z and x-z plane, respectively) in the z, x and y-direction, respectively. This way, the block is free to have lateral contraction. At the back surface the displacement is prescribed in several increments. The mesh with boundary conditions are shown on the left in figure 5.3.2.4.1, while on the right in the same figure the deformation of this mesh is shown with contour-bands of the y-displacements.

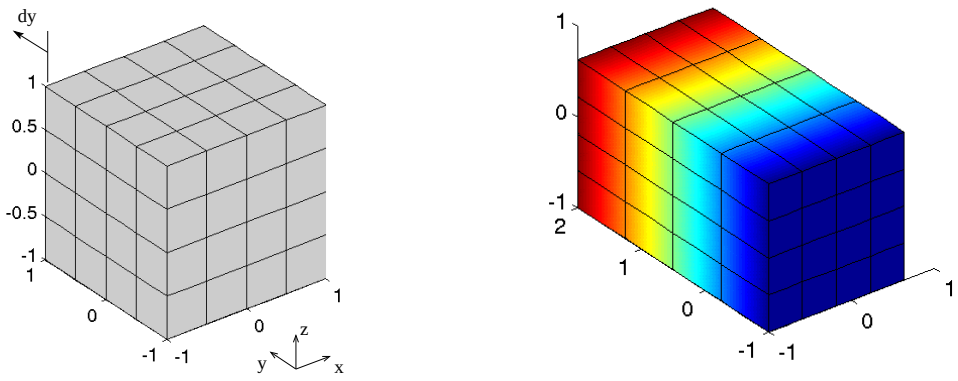


Figure 5.3.2.4.1: Mesh and boundary conditions (left) and deformed block with y-displacement contour-bands (right)

In the next pages the files can be found.

Mesh file

```
#  block3d.msh
#
#  mesh input file for a 3D block of dimensions 1 x 1 x 1
#  geometrically non-linear deformation of a 3D block
#  A uniaxial tension test enforced by prescribing the displacement
#  along one surface.
#
#  The updated Lagrange approach is used
#  This includes a pseudo time loop, with non-linear iteration per step
#
#  See Manual Standard Elements Section 5.3.2
#  and examples manual Section 5.3.2.4
#
#  To create the mesh run:
#
#  sepmesh block3d.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#

constants           # See Users Manual Section 1.4

   integers

    n = 4               # number of elements in x-direction
    m = 4               # number of elements in y-direction
    l = 4               # number of elements in z-direction
    lin = 2             # quadratic line elements
    sur = 6             # bi-quadratic quadrilaterals
    vol =14             # tri-quadratic hexahedrons

  reals
    xl = 1.0            # length of block in x-dir
    yl = 1.0            # length of block in y-dir
    zl = 1.0            # length of block in z-dir

end
#
#  Define the mesh
#
mesh3d            # See Users Manual Section 2.2
#
#  user points
#
   points             # See Users Manual Section 2.2

    p1=(- xl,- yl,- zl)        # Left under point bottom surface
    p2=( xl,- yl,- zl)         # Right under point bottom surface
    p3=( xl, yl,- zl)          # Right upper point bottom surface
    p4=(- xl, yl,- zl)         # Left upper point bottom surface
    p5=(- xl,- yl, zl)         # Left under point top surface
```

```
    p6=( xl,- yl, zl)          # Right under point top surface
    p7=( xl, yl, zl)           # Right upper point top surface
    p8=(- xl, yl, zl)          # Left upper point top surface
#
#   curves
#
   curves          # See Users Manual Section 2.3

    #curves of bottom surface
    c1  = line  lin ( p1,p2,nelm= n)
    c2  = line  lin ( p2,p3,nelm= m)
    c3  = line  lin ( p3,p4,nelm= n)
    c4  = line  lin ( p4,p1,nelm= m)
    #curves of top surface
    c5  = line  lin ( p5,p6,nelm= n)
    c6  = line  lin ( p6,p7,nelm= m)
    c7  = line  lin ( p7,p8,nelm= n)
    c8  = line  lin ( p8,p5,nelm= m)
    #curves determining the height of the block
    c9  = line  lin ( p1,p5,nelm= l)
    c10 = line  lin ( p2,p6,nelm= l)
    c11 = line  lin ( p3,p7,nelm= l)
    c12 = line  lin ( p4,p8,nelm= l)

#
#  surface
#
   surfaces       # See Users Manual Section 2.4

    s1 = rectangle  sur (c1,c2,c3,c4)         # bottom surface
    s2 = rectangle  sur (c1,c10,-c5,-c9 )     # front surface
    s3 = rectangle  sur (c2,c11,-c6,-c10)     # right surface
    s4 = rectangle  sur (-c3,c11,c7,-c12)     # back surface
    s5 = rectangle  sur (-c4,c12,c8,-c9 )     # left surface
    s6 = rectangle  sur (c5,c6,c7,c8)         # top surface
#
#  volume
#

   volumes       # See Users Manual Section 2.5

    v1 = brick  vol (s1,s2,s3,s4,s5,s6)


#  Plot of mesh

   plot, eyepoint = (50, 20, 20)

end
```

Problem definition file

```
#  block3d.prb
#
#  geometrically non-linear deformation of a 3D block
#  A uniaxial tension test enforced by prescribing the displacement
#  along one surface.
#
#  The updated Lagrange approach is used
#  This includes a pseudo time loop, with non-linear iteration per step.
#
#  See Manual Standard Elements Section 5.3.2
#  and examples manual Section 5.3.2.4
#
#  To run this file use:
#     sepcomp block3d.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#

#  Suppress superfluous output

set warn off
set output none

#
#  Define some general constants
#

constants           # See Users Manual Section 1.4

   integers
      nincr = 10           # number of artificial time steps
   reals
      tstart = 0           # start of artificial time algorithm
      tend = 0.5           # end of artificial time algorithm
      dt =   tend/ nincr   # artificial time step defined by tend/nincr
   variables
      incr = 0             # counter for increments (used for printing only)
   vector_names
    #   The following vectors are used for the computation of the displacement
    #   Mark that the vectors u and un must always be given
    #   as first and second vector
      u          # Displacement vector per pseudo time step
      un         # Total displacement vector
    #   Output vectors
      stress     # stress
      strain     # strain
      pressure   # pressure
end


problem                   # See Users Manual Section 3.2.2
```

```
   types                          # Define types of elements,
                                  # See Users Manual Section 3.2.2

      elgrp1 = 201              # element type for solid material
                                # updated Lagrange approach
                                # Crouzeix-Raviart elements (discontinuous pressure)

   essbouncond                    # Define where essential boundary conditions are
                                  # given (not the value)
                                  # See Users Manual Section 3.2.2
      degfd1 = surfaces(s5)     #left surface
      degfd2 = surfaces(s2)     #front surface
      degfd3 = surfaces(s1)     #bottom surface
      degfd2 = surfaces(s4)     #back surface
  ### Crouzeix-Raviart (201)
   renumber levels (1,2,3),(4,5,6,7)  # renumbering of unknowns per level
                                  # first displacements, then pressures
                                  # in this way zero pivots are avoided
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   storage_method = compact   # Non-symmetrical compact matrix
               # So an iterative solver is applied
end

# define essential boundary conditions
# no extra input since all are equal to zero
#  See Users Manual Section 3.2.5

essential boundary conditions
    surfaces(s4), degfd2, value = 0.1  # Displacing back surface in positive
                                       # y-direction each increment
end

#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 5.3.2

coefficients
  # internal elements
   elgrp1 (nparm = 45)      # The coefficients are defined by 45 parameters
      icoef2 = 0            # type of stress-strain relation
                            # 0 - full 3D
      icoef3 = 0            # type of numerical integration
                            # 0 - default value
      icoef4 = 2            # constitutive law
                            # 2  - incompressible Neo-Hookean
      icoef5 = 0            # user flags, coef = iusrvec + 100*iusrflg
                            # iusrvc = 0 - user vector is not filled
      coef10 = 3d3
                            # Mooney-Rivlin: material parameter c0
```

```
end
#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#
structure

   ### Create total displacement vector   and set to 0
   create_vector,  un

   ### Start incremental (pseudo-time) loop
   start_time_loop

      ### Clear solution vector (Displacement vector per pseudo time step)
      create_vector,  u

      time_integration  # Adjust the time parameters
                                          # No actual action
                                          # No actual action

      incr = incr+1                       # Raise increment counter

      ### Print time and increment number
      print_time
      print incr, text = 'increment'

      ### Solve system of non-linear equations to get new increment vector
      solve_nonlinear_system,  u

      ### Compute stress, strain and pressure vectors
      #   This must be done before updating the mesh and un

      derivatives, seq_deriv = 1,  stress
      derivatives, seq_deriv = 2,  strain
      derivatives, seq_deriv = 3,  pressure

      ### Deform mesh using the displacement vector
      deform_mesh,  u

      ### Update total solution vector
      un =  un +  u

   ### End time loop
   end_time_loop

end

#  Definition of (pseudo) time integration
#  See Users Manual Section 3.2.15

time_integration
   method = stationary    # no action, just adjusting time parameters
   tinit =  tstart        # start time
   tend =  tend           # end time
   tstep =  dt            # time step
```

```
end

# Definition of iteration for non linear equations
#  See Users Manual Section 3.2.9

nonlinear_equations
   global_options, maxiter = 50, miniter = 1, accuracy = 1d-3//
      criterion = relative,  print_level = 2,  at_error= return //
      iteration_method = newton
   equations 1
     fill_coefficients = 1
end

#  Compute stress tensor
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives
  icheld = 6                    # compute stress
  seq_input_vector =  u         # use the total displacement as input vector
end

#  Compute stress tensor
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives, sequence_number = 2
  icheld = 8                    # compute strain
  seq_input_vector =  u         # use the total displacement as input vector
end

#  Compute pressure
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives, sequence_number = 3
  icheld = 7                    # compute pressure
  seq_input_vector =  u         # use the total displacement as input vector
end

#  Information for linear solver
#  See Users Manual Section 3.2.8

solve
    iterative_method = BICGSTAB, preconditioner = ilu
end
```

5.3.2.5 Arterial wall with internal pressure (3D)

This example treats the deformation of a 3D arterial wall by an internal pressure. The solid material is described by a neo-Hookean incompressible material law as described in the manual Standard Problems Section 5.3.2 (element type 202). 27-noded hexahedrons are used for the mesh (shape 14). Making use of symmetry, 1/4th of the arterial wall is meshed as shown at the left side of figure 5.3.2.5.1. The pressure is applied at the inner wall of the artery and is increased incrementally. The geometry and pressure contour-bands after deformation are shown on the right of figure 5.3.2.5.1.

To get this example into your local directory use:

```
sepgetex artery3d
```

and to run it use:

```
sepmesh artery3d.msh
sepcomp artery3d.prb
```

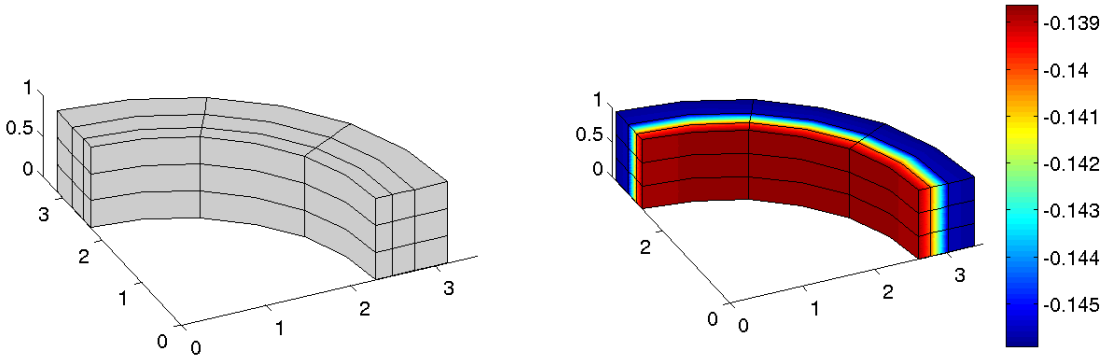


Figure 5.3.2.5.1: Mesh (left) and corresponding deformations with pressure contour bands (right)

Exactly the same method as in Section 5.3.2.1 is applied.

In the next pages the files can be found.

Mesh file

```
#  artery3d.msh
#
#  mesh input file for
#  geometrically non-linear deformation of a 3D artery wall
#  A pressure is applied at the inner arterial wall.
#  Only 1/4-th of the region is used
#
#  The updated Lagrange approach is used
#  This includes a pseudo time loop, with non-linear iteration per step
#  The pressure is increased during the time stepping
#
#  See Examples Manual Section 5.3.2.5
#
#  To create the mesh run:
#
#  sepmesh artery3d.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#

constants           # See Users Manual Section 1.4

   integers
      linetype    = 2                 # quadratic line elements
      surftype    = 6                 # bi-quadratic quadrilaterals
      voltype     = 14                # tri-quadratic hexahedrons

      n_r         = 3          # number of elements over tube radius
      n_len       = 3          # number of elements along the tube length
      n_arc       = 3          # number of elements along the wall

   reals
      ri          = 2.3        # inner radius
      wt          = 0.85       # wall thickness
      len         = 1.0        # length of the tube

      mshfac      = 2          # mesh factor

      arc         =  (90/360)*2* pi     # angle of 90 degrees, expressed in radians

      # constants computed in COMPCONS:
      r1 =   ri+ wt                 # r1 = ri + wt (outer radius)
      x3 =   ri* cos( arc)          # x-coordinate of end point
      y3 =   ri* sin( arc)          # y-coordinate of end point
      x4 =   r1* cos( arc)          # x-coordinate of end point
      y4 =   r1* sin( arc)          # y-coordinate of end point
end
#
#  Define the mesh
#
mesh3d            # See Users Manual Section 2.2
```

```
#
#  user points
#
   points             # See Users Manual Section 2.2
      p99  = (      0,      0,      0)     # Centre
      p1   = (     ri,      0,      0)
      p2   = (     r1,      0,      0)
      p3   = (     x3,     y3,      0)
      p4   = (     x4,     y4,      0)
      p9   = (     ri,      0,    len)
      p10  = (     r1,      0,    len)
      p11  = (     x3,     y3,    len)
      p12  = (     x4,     y4,    len)
#
#   curves
#
   curves          # See Users Manual Section 2.3
      c1 = line linetype (p1, p2, nelm =  n_r, ratio = 1, factor =  mshfac)
      c2 = arc linetype(p1, p3, p99, nelm= n_arc)
      c3 = line linetype(p3, p4, nelm =  n_r, ratio = 1, factor =  mshfac)
      c4 = arc linetype(p4, p2, p99, nelm= n_arc)

      # front to back connection
      c5 = line linetype (p1, p9,  nelm =  n_len) #inner
      c6 = line linetype (p2, p10, nelm =  n_len) #outer
      c7 = line linetype (p3, p11, nelm =  n_len) #inner
      c8 = line linetype (p4, p12, nelm =  n_len) #outer

      # translate into the back
      c9  = translate c1 (p9, p10)
      c10 = translate c2 (p9,p11) #inner
      c11 = translate c3 (p11, p12)
      c12 = translate c4 (p12,p10) #outer

#
#  surface
#
   surfaces       # See Users Manual Section 2.4

      s1 = quadrilateral surftype (-c1, c2, c3, c4, curvature = 2) #front
      s2 = translate s1 (-c9, c10, c11, c12)                       #back
      s3 = pipesurface surftype (c2, c10, c5, c7)                  #inner
      s4 = pipesurface surftype (c4, c12, c8, c6)                  #outer
      s5 = pipesurface surftype (-c1, -c9, c6, c5)                 #horz
      s6 = pipesurface surftype (c3, c11, c7, c8)                  #vert
      s7 = ordered surface (s5, s3, s6, s4)
#
#  volume
#

   volumes       # See Users Manual Section 2.5
      v1 = pipe voltype (s1, s2, s7)

#  renumbering of nodes
```

```
   renumber best, levels

#  Plot of mesh

   plot, curve = 2, eyepoint = (50, 20, 20), rotate = 1

end
```

Problem definition file

```
#  artery3d.prb
#
#  geometrically non-linear deformation of an arterial wall.
#  Using symmetry, 1/4th of the wall is modeled.
#  An internal pressure is applied at the inner arterial wall.
#
#  The updated Lagrange approach is used
#  This includes a pseudo time loop, with non-linear iteration per step.
#  The axial length is constrained by fixing the begin and end surface.
#  The pressure is increased during the time stepping
#
#  See Manual Standard Elements Section 5.3.2
#  and examples manual Section 5.3.2.5
#
#  To run this file use:
#     sepcomp artery3d.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#

#  Suppress superfluous output

set warn off
set output none

#
#  Define some general constants
#

constants           # See Users Manual Section 1.4

   integers
      nincr = 10           # number of artificial time steps
   reals
      tstart = 0           # start of artificial time algorithm
      tend = 1             # end of artificial time algorithm
      dt =   tend/ nincr   # artificial time step defined by tend/nincr
   variables
      incr = 0             # counter for increments (used for printing only)
      force = 0            # is used to define pressure at inner wall
   vector_names
    #   The following vectors are used for the computation of the displacement
    #   Mark that the vectors u and un must always be given
    #   as first and second vector
      u          # Displacement vector per pseudo time step
      un         # Total displacement vector
    #   Output vectors
      stress     # stress
      strain     # strain
      pressure   # pressure
end
```

```
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2

      elgrp1 = (type = 202)    # element type for solid material
                               # updated Lagrange approach
                               # Taylor-Hood elements (continuous pressure)
   natbouncond             # Definition of type numbers for natural
                           # boundary conditions
      bngrp1 = 210         # boundary group used to apply internal pressure

   bounelements            # Definition of boundary elements
      belm1 = surfaces (s3)     # inner surface

   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      degfd1 = surfaces(s6)     # symmetry plane in y-z plane
      degfd2 = surfaces(s5)     # symmetry plane in x-z plane
      degfd3 = surfaces(s1)     # bottom
      degfd3 = surfaces(s2)     # top
    renumber levels (1,2,3),(4)  # renumbering of unknowns per level
                               # first displacements, then pressures
                               # in this way zero pivots are avoided
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   storage_method = compact   # Non-symmetrical compact matrix
               # So an iterative solver is applied
end

#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 5.3.2

coefficients
  # internal elements
   elgrp1 (nparm = 45)      # The coefficients are defined by 45 parameters
      icoef2 = 0            # type of stress-strain relation
                            # 0 - full 3D
      icoef3 = 0            # type of numerical integration
                            # 0 - default value
      icoef4 = 2            # constitutive law
                            # 2  - incompressible Neo-Hookean
      icoef5 = 0            # user flags
                            # iusrvc = 0 - user vector is not filled
      coef7 = 1             # Take into account the linearization of
                            # the Jacobian
      coef10 = 1            # shear modulus
```

```
  # boundary elements
   bngrp1 (nparm = 25)      # The coefficients are defined by 25 parameters
      icoef1= 2             # 2 - local coor system with linearization for
                            # boundary conditions
      icoef3=1              # Integration rule (1=Newton Cotes)
      coef6=  force         # force in normal direction as a function (p=0.1 t)

end
#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#
structure

   ### Create total displacement vector   and set to 0
   create_vector,  un

   ### Start incremental (pseudo-time) loop
   start_time_loop

      ### Clear solution vector (Displacement vector per pseudo time step)
      create_vector,  u

      time_integration  # Adjust the time parameters
                                          # No actual action
                                          # No actual action

      incr = incr+1                       # Raise increment counter
                                          # t = incr*dt
      force = 0.1*incr*dt                 # Compute force as function of t

      ### Print time and increment number
      print_time
      print incr, text = 'increment'

      ### Solve system of non-linear equations to get new increment vector
      solve_nonlinear_system,  u

      ### Compute stress, strain and pressure vectors
      #   This must be done before updating the mesh and un

      derivatives, seq_deriv = 1,  stress
      derivatives, seq_deriv = 2,  strain
      derivatives, seq_deriv = 3,  pressure

      ### Deform mesh
      deform_mesh,  u

      ### Update total solution vector
      un =  un +  u

   ### End time loop
   end_time_loop

end
```

```
#  Definition of (pseudo) time integration
#  See Users Manual Section 3.2.15

time_integration
   method = stationary    # no action, just adjusting time parameters
   tinit =  tstart        # start time
   tend =  tend           # end time
   tstep =  dt            # time step
end

# Definition of iteration for non linear equations
#  See Users Manual Section 3.2.9

nonlinear_equations
   global_options, maxiter = 50, miniter = 1, accuracy = 1d-3//
      criterion = relative,  print_level = 2,  at_error= return //
      iteration_method = newton
   equations 1
     fill_coefficients = 1
end

#  Compute stress tensor
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives
  icheld = 6                    # compute stress
  seq_input_vector =  u         # use the total displacement as input vector
end

#  Compute stress tensor
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives, sequence_number = 2
  icheld = 8                    # compute strain
  seq_input_vector =  u         # use the total displacement as input vector
end

#  Compute pressure
#  See Users Manual Section 3.2.11 and Standard problems Section 5.3.2

derivatives, sequence_number = 3
  icheld = 7                    # compute pressure
  seq_input_vector =  u         # use the total displacement as input vector
end

#  Information for linear solver
#  See Users Manual Section 3.2.8

solve
    iterative_method = BICGSTAB, preconditioner = ilu
end
```

5.4 (Thick) plate elements

5.4.1 Some analytical tests for the plate elements

In order to test the plate elements we compare the numerical solution with some simple examples of which the analytical solution is known. It concerns the uniform load on three types of plates:

- A circular plate (radius 5)
- A rectangular plate (Size 10 × 20)
- A square plate (Size 10)

Both case of a clamped plate and a plate of which the boundary is simply supported are investigated. These examples can be found in Hughes (1987). Because of symmetry it is in all cases sufficient to consider only one quarter of the region. In order to get these examples into your local directory use the command `sepgetex`. The following files are available with sepgetex:

```
sepgetex circplatecl      (Circular plate clamped)
sepgetex circplatess      (Circular plate simply supported)
sepgetex rectplatecl      (Rectangular plate clamped)
sepgetex rectplatess      (Rectangular plate simply supported)
sepgetex squaplatecl      (Square plate clamped)
sepgetex squaplatess      (Square plate simply supported)
```

Comparison with the analytical results shows a good convergence behaviour when the mesh is refined. Table 5.4.1.0.1 compares the analytical solution in the centre of the plate with the numerical one for various mesh sizes. Mark that the results on the circular plate can not be compared with those of Hughes, since the meshes are different.

Table 5.4.1.0.1 Accuracy of the plate elements

Type of plate	number of elms	analytical	numerical
Circular	2x2	0.097656	0.0867866
Clamped	4x6		0.0939645
	10x16		0.0964283
Circular	2x2	0.398137	0.344108
Simply	4x6		0.384165
supported	10x16		0.394718
Rectangular	2x2	0.260073	0.251341
Clamped	4x4		0.247073
	8x8		0.251917
	32x32		0.253550
Rectangular	2x2	1.016484	0.998857
Simply	4x4		1.00748
supported	8x8		1.01221
	32x32		1.01508
Square	2x2	0.126374	0.121342
Clamped	4x4		0.125315
	8x8		0.126414
	32x32		0.126762
Square	2x2	0.406593	0.397278
Simply	4x4		0.404656
supported	8x8		0.406530
	32x32		0.408408

Some of the corresponding input files are given below without extra text, except the comments that can be found in the input files. First we consider `circplatecl.msh`

```
#  circplatecl.msh
#  Test problem for the plate elements
#  Circular plate, uniform load, clamped edge
#  Only a quarter of the plate is computed
#
#  See Manual Standard Elements Section 5.4.1
#  and examples manual Section 5.4.1
#
#  To run this file use:
#     sepmesh circplatecl.msh
#
#  Creates the file meshoutput
#
#  Define some constants
#
constants           # See Users Manual Section 1.4
   integers
      na = 4        # number of elements along the radius
      nb = 6        # number of elements along the arc
   reals
      radius = 5    # Radius of circle
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points                 # See Users Manual Section 2.2
      p1 = ( 0, 0)        # Centre of circle
      p2 = (  radius, 0)  # At most left point
      p3 = ( 0,  radius)  # At most upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1 = line 1 (p1, p2, nelm =  na)      # straight horizontal line
      c2 =  arc 1 (p2, p3, p1, nelm =  nb)  # quarter of circle
      c3 = line 1 (p3, p1, nelm =  na)      # straight vertical line
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1 = general 5 (c1, c2, c3)

   plot                          # make a plot of the mesh
                                 # See Users Manual Section 2.2
end
```

The corresponding problem file `circplatecl.prb` is given by:

```
# circplatecl.prb
#
#  problem file for the plate elements
#  Circular plate, uniform load, clamped edge
#  Only a quarter of the plate is computed
#
#  See Manual Standard Elements Section 5.4.1
#  and examples manual Section 5.4.1
#
#  To run this file use:
#     sepcomp circplatecl.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#  Define some constants
#
constants           # See Users Manual Section 1.4
   reals
      E    = 10.92e5  # Young's modulus
      nu   = 0.3      # Poisson's ratio
      h    = 0.1      # thickness of the plate
      load = 1        # distributed load in z-direction
   vector_names
      displacement
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
      elgrp1 = (type = 255)     # Type number for plate elements
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
      curves (c2)               # clamped edge (w=0, theta = 0)

      degfd2, curves (c1)       # symmetry edge (Theta_1 = 0 )
      degfd3, curves (c3)       # symmetry edge (Theta_2 = 0 )
end
#
#  Define the structure of the main program
#
structure                   # See Users Manual Section 3.2.3
   prescribe_boundary_conditions displacement
   solve_linear_system displacement
   print displacement, points(p1)   # print the solution in the centre of the plate
end

#  Define the coefficients for the problems
#  See Users Manual Section 3.2.6
#  See also standard problems Section 5.4

coefficients
   elgrp1 (nparm = 45)      # coefficients for plate elements
```

```
      icoef 2 = 0           # Isotropic material
      coef 6  =  E          # Young's modulus
      coef 7  =  nu         # Poisson's ratio
      coef 27 =  h          # thickness of the plate
      coef 28 =  load       # distributed load in z-direction
end

end_of_sepran_input
```

The file `circplatess.msh` is identical to `circplatecl.msh` and will not be repeated.
The corresponding file `circplatess.prb` differs a little bit from `circplatecl.prb`.
In fact only the part under the keyword `problem`, subkeyword `essbouncond` is different. This part is printed only.

```
   essbouncond                # Define where essential boundary conditions are
                              # given (not the value)
      degfd1, curves (c2)     # simply supported edge (w=0)

      degfd2, curves (c1)     # symmetry edge (Theta_1 = 0 )
      degfd3, curves (c3)     # symmetry edge (Theta_2 = 0 )
end
```

The file `rectplatecl.msh` is given below.

```
#  rectplatecl.msh
#  Test problem for the plate elements
#  Rectangular plate (10x20), uniform load, clamped edge
#  Only a quarter of the plate is computed
#
#  See Manual Standard Elements Section 5.4.1
#  and examples manual Section 5.4.1
#
#  To run this file use:
#     sepmesh rectplatecl.msh
#
#  Creates the file meshoutput
#
#  Define some constants
#
constants           # See Users Manual Section 1.4
   integers
      ne = 8        # number of elements along a side
   reals
      L =  5        # Half length of plate
      H = 10        # Half height of plate
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points            # See Users Manual Section 2.2
      p1 = ( 0, 0)   # Point left under
      p2 = (  L, 0)  # Point right under
      p3 = (  L,  H) # Point left upper
      p4 = ( 0,  H)  # Point right upper
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1 = line 1 (p1, p2, nelm =  ne)
      c2 = line 1 (p2, p3, nelm =  ne)
      c3 = line 1 (p3, p4, nelm =  ne)
      c4 = line 1 (p4, p1, nelm =  ne)
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1 = rectangle 5 (c1, c2, c3, c4)

   plot                          # make a plot of the mesh
                                 # See Users Manual Section 2.2
end
```

The other files are not repeated here. If you want to investigate them, use the command sepgetex.

5.5 Contact problems

In this Chapter we demonstrate a number of contact problems Presently the following examples are available:

The Hertz problem (5.5.1) An infinitely long, elastic, half cylinder is pressed on a flat surface.

The Roll problem (5.5.2) A cylinder is pressed between two flat surfaces.

The Wheel problem (5.5.3) A tire fixed on a hub is pressed downwards on the ground.

5.5.1 The Hertz problem

In this example we consider a very simple example of a contact problem. Consider an infinitely long half cylinder that is pressed onto a flat surface. Since the cylinder is pressed downwards it deforms and displacement is directed downwards. However, the flat surface prevents the cylinder to move below the plane. To analyze this problem it is sufficient to consider only a slice of the half cylinder, due to symmetry in the length direction.
To get this example into your local directory use:

```
sepgetex hertz
```

To run the problem use

```
seplink hertz
hertz < hertz.prb
seppost hertz.pst
```

The shape of the slice can be seen easily by the plot of the curves in Figure 5.5.1.1. The flat surface

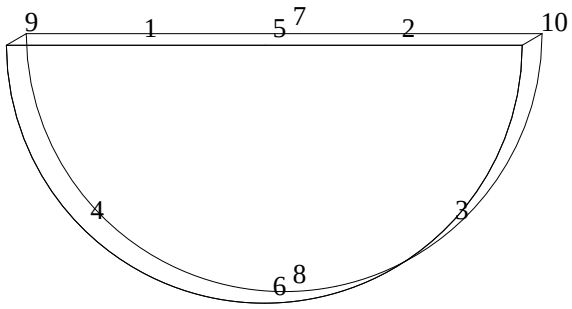


Figure 5.5.1.1: Definition of the curves in the slice

is represented by the plane $z = 0$. On the top of the half cylinder we prescribe a fixed displacement downwards, which represents the downwards pressing. The cylinder is supposed to be linearly elastic with Poisson's ratio $\nu = 0.3$. The elasticity modulus E is not important for this problem so we take the value $E = 1$. The free parts of the cylinder are stress free. Since the cylinder is pressed onto the flat surface, contact is made. On those places where we have contact the position of the cylinder must be equal to $z = 0$. Since this is a contact problem it is essentially non-linear.
The following algorithm is applied to compute the contact surface:
In each step of the algorithm the contact surface is computed. The contact surface is defined as the set of points with z coordinate less than or equal to 0. Hence all points that are on or below the flat surface $z = 0$. Since the contact surface is not known a priori this surface may change in each iteration. Not only is it possible that points are added to the contact surface also they may be removed from the contact surface. This is the case if the reaction force in the contact surface points is pointed upwards, i.e. the third component of the reaction force is positive.
Once the contact surface is computed the displacement of the z-component of these point is set equal to $-z$. Hence the sum of the displacement and the original position of the surface is precisely equal to 0. The linear elasticity problem is solved and the reaction force computed.
Next the contact distance is computed as the sum of the z-component of the original cylinder and the z-component of the displacement. Hence a negative contact distance means that there is contact. Also the contact force is computed, which is actually the third component of the reaction

force.
This process is repeated until convergence is achieved.
In a scheme the contact algorithm can be written as

Create the mesh
Initialize all vectors
while not converged **do**
  Compute the contact surface
  Store the essential boundary conditions into the displacement vector
  Solve the linear elasticity problem and compute the reaction force
  Compute the contact distance and the contact force
end while

Due to the symmetry in length direction it is sufficient to take only one row of elements in that direction. Since we expect that the displacement changes the most in the neighbourhood of the contact surface, the region is refined in the neighbourhood of the plane $z = 0$. The following input file may be used to solve the hertz problem. It contains both a description of the mesh and the problem file.

```
#  hertz.prb
#
#  Hertz-problem:
#   An infinitely long, elastic, half cylinder is pressed
#     on a flat surface.
#   A slice of this cylinder is analyzed.
#  See Manual Examples Section 5.5.1
#
#  To run this file use:
#     seplink hertz
#     hertz < hertz.prb
#
#  Creates the files meshoutput and sepcomp.out
#
#  Define some general constants
#
constants
   vector_names   # names of vectors to be used in the computation
      displacement         # displacement_vector
      reaction_force       # vector with reaction_forces
      contact_distance     # vector in which the contact distance is stored
      contact_force        # vector in which the contact force is stored
      stress               # stress tensor
      strain               # strain tensor
end
#
#  Some information at the start of the program
#
start             # See Users Manual Section 3.2.1
   norotate           # Plots may not be rotated
end
#
#  First we define the mesh in the slice
#
mesh3d             # See Users Manual Section 2.2
  coarse (unit = 0.1)               # define the unit length of elements
                                    # In the contact region at the bottom the
                                    # mesh is refined
```

```
#
#  user points
#
   points           # See Users Manual Section 2.2
    p1 = ( 0.0, 0.0, 1.0, 1.00)     # centre point at front side of top of
                                    # cylinder
    p2 = ( 1.0, 0.0, 1.0, 1.00)     # right-hand side point at front side of top
                                    # of cylinder
    p3 = (-1.0, 0.0, 1.0, 1.00)     # left-hand side point at front side of top
                                    # of cylinder
    p4 = ( 0.0, 0.0, 0.0, 0.25)     # bottom point of front side of cylinder

    p5 = ( 0.0, 0.1, 1.0, 1.00)     # centre point at back side of top of
                                    # cylinder
    p6 = ( 1.0, 0.1, 1.0, 1.00)     # right-hand side point at back side of top
                                    # of cylinder
    p7 = (-1.0, 0.1, 1.0, 1.00)     # left-hand side point at back side of top
                                    # of cylinder
    p8 = ( 0.0, 0.1, 0.0, 0.25)     # bottom point of back side of cylinder
#
#  curves
#
   curves           # See Users Manual Section 2.3
    c1 = cline 1 (p3, p1)           # Line  at front side of top of
                                    # cylinder from left to centre
    c2 = cline 1 (p1, p2)           # Line  at front side of top of
                                    # cylinder from centre to right
    c3 =  carc 1 (p2, p4, p1)       # Right-hand side part of curved part of
                                    # front side of cylinder
    c4 =  carc 1 (p4, p3, p1)       # Left-hand side part of curved part of
                                    # front side of cylinder
    c5 = curves (c1, c2)            # Top of half cylinder (front side)
    c6 = curves (c3, c4)            # Curved part of half cylinder (back side)
    c7 = translate c5 (p7, p5, p6)  # Top of half cylinder (back side)
    c8 = translate c6 (p6, p8, p7)  # Curved part of half cylinder (front side)
    c9 = line 1 (p3, p7, nelm = 1)  # Line from front side to back side the
                                    # left
    c10 = line 1 (p2, p6, nelm = 1) # Line from front side to back side the
                                    # right
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
    s1 = general 5 (c5, c6)              # front end of half cylinder
    s2 = translate s1 (c7, c8)           # back end of half cylinder
    s3 = pipesurface 5 (c5, c7, c9, c10)  # top of half cylinder
    s4 = pipesurface 5 (c6, c8, c10, c9)  # curved envelope of half cylinder
    s5 = ordered surface ((s3,s4))       # total envelope of half cylinder
#
#  volumes
#
  volumes         # See Users Manual Section 2.5
    v1 = pipe 13 (s1, s2, s5)            # Complete half cylinder

  plot, eyepoint = (2.0, -3.0, 2.0)      # make a plot of the mesh
```

```
                                        # See Users Manual Section 2.2
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                          # Define types of elements,
                                  # See Users Manual Section 3.2.2
      elgrp 1 = (type=250)        # Type number for linear elasticity
                                  # See Standard problems Section 5.1
   essbouncond                    # Define where essential boundary conditions are
                                  # given (not the value)
                                  # See Users Manual Section 3.2.2
      degfd 2, surfaces (s1)      # No displacement in y-direction of front end
      degfd 2, surfaces (s2)      # No displacement in y-direction of back end
      surfaces (s3)               # Prescribed displacement in top of half
                                  # cylinder
      degfd 3, contact 1          # The z-displacement is -z in contact points
end
#
#   Input for the contact algorithm
#
contact, sequence_number = 1     # See Users Manual Section 3.2.16
  contact_surface = s4                      # surface that makes contact
  contact_distance = contact_distance     # vector to be used to store the
                                            # contact distance
  contact_force = contact_force           # vector to be used to store the
                                            # contact force
  contact_method = NEG_DISTANCE             # defines when a point is supposed
                                            # to make contact (in this case
                                            # if the contact distance < 0)
  contact_disable_method = CONTACT_FORCE    # defines when a point is supposed
                                            # to lose contact (in this case
                                            # if the contact force < 0)
end
#
#  Define non-zero essential boundary conditions
#  See Users Manual Section 3.2.5
#
essential boundary conditions, sequence_number = 1
  degfd 3, surfaces (s3), value = -0.2      # The displacement in z-direction of
                                            # the top surface = -0.2
  degfd 3, contact 1, func = 1          # In those points where we have contact
                                        # the displacement is made equal to -z,
                                        # so that the points are moved back to
                                        # z = 0
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because we have a free boundary problem

structure                  # See Users Manual Section 3.2.3
```

```
  write_mesh     # First the mesh is written to the file meshoutput
                 # in order to be used for postprocessing

  # Next create 4 vectors that are used during the analysis
  # The displacement vector and the reaction force vector are set equal to 0
  # They contain 3 degrees of freedom per point
  create_vector, sequence_number = 1, displacement
  create_vector, sequence_number = 1, reaction_force

  # The vectors contact_distance and contact_force contain one degree of
  # freedom per unknown and are also initialized to 0
  create_vector, sequence_number = 2, contact_distance
  create_vector, sequence_number = 2, contact_force

  # In order to solve the (non-linear) contact problem we define a
  # loop by start_loop ... end_loop

  start_loop, sequence_number = 1
   # Compute the contact surface using the input for the contact problem
    compute_contact_surface, sequence_number = 1

   # Store the essential boundary conditions in the displacement vector
   # Since they depend on the contact surface they may change in each step
    prescribe_boundary_conditions, sequence_number = 1, displacement

   # Solve the displacement vector by the linear elasticity problem
   # Compute the reaction force vector, necessary for the contact
   # algorithm
    solve_linear_system, //
        seq_solve = 1, seq_coef = 1, displacement//
          reaction_force = reaction_force

   # Recompute the contact distance and the contact force
    create_vector, sequence_number = 3, contact_distance
    create_vector, sequence_number = 4, contact_force

  end_loop


  # Finally compute the stress and the strain tensors
  derivatives, seq_deriv = 1, seq_coef = 1, stress
  derivatives, seq_deriv = 2, seq_coef = 1, strain

  output
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
  storage_scheme = compact, symmetric, reaction_force
                   # symmetrical matrix with compact storage
                   # hence an iterative linear solver is used
                   # reaction forces must be computed
end
```

```
#  Input for the loop in the structure block
#  Defines how many iterations may be carried out at most
#  and when the process is finished
#  See Users Manual Section 3.2.3

loop_input, sequence_number = 1
  maxiter = 50                    # maximum number of iterations
  miniter = 2                     # minimum number of iterations
  accuracy = 1d-4                 # relative accuracy
  criterion = relative
  seq_vector = contact_distance   # vector to be used to check the convergence
  print_level = 2                 # defines the amount of output
end

# Input for the linear solver
#  See Users Manual Section 3.2.8

solve, sequence_number = 1
  iteration_method = cg, //
      start=old_solution, //
      preconditioning=ilu, //
      accuracy = 0.01
end

#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 5.1

coefficients, sequence_number = 1
  elgrp 1 (nparm=45)             # The coefficients are defined by 45 parameters
    icoef 2 = 0                  # type of stress-strain relation
                                 # 0: plane stress
     coef 6 = 1.0                # Elasticity modulus
     coef 7 = 0.3                # Poisson ratio
end

# Create start vectors
#  See Users Manual Section 3.2.10
#  First displacement and reaction force
#  Type solution vector

create vector, sequence_number = 1
  value = 0
end

#  Next contact_distance and contact_force
#  One degree of freedom per point

create vector, sequence_number = 2
  type = vector of special structure v1
  value = 0
end

# Create contact_distance during the iterations
```

```
# The contact distance is defined as the sum of the z-displacement and
# the z coordinate
# The summation is carried out in subroutine funcvect

create vector, sequence_number = 3
  type = vector of special structure v1
  surfaces (s4), old_vector = contact_distance, seq_vectors = displacement
end

# Create contact_force during the iterations
# The contact force is equal to the third component of the reaction force
# The extraction is carried out in subroutine funcvect

create vector, sequence_number = 4
  type = vector of special structure v1
  surfaces (s4), old_vector = contact_force, seq_vectors = reaction_force
end

# compute stress
# See Users Manual, Section 3.2.11 and Standard problems Section 5.1

derivatives, sequence_number = 1
  icheld = 6
end

# compute strain
# See Users Manual, Section 3.2.11 and Standard problems Section 5.1

derivatives, sequence_number = 2
  icheld = 7
end

# write the results to the file sepcomp.out
# See Users Manual, Section 3.2.13

output
end
```

Figure 5.5.1.2 shows the mesh used in this problem This file requires a main program with subroutines, since the boundary condition in the contact surface depends on space and in order to compute the contact distance and contact force. Th main program used by us is:

```
      program hertz

!      --- Main program for the Hertz-problem:
!          An infinitely long, elastic, half cylinder is pressed
!          on a flat surface.
!          A slice of this cylinder is analyzed.
!          This main program is necessary because of the variable boundary
!          conditions
      implicit none

      integer, allocatable, dimension (:) :: ibuffr
      integer pbuffr, error
      parameter ( pbuffr=25000000)
```

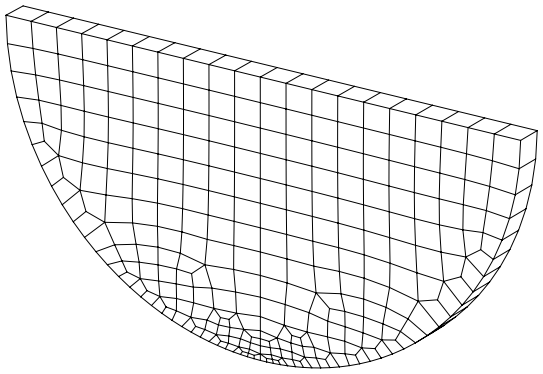


Figure 5.5.1.2: Mesh created in the slice

```
      allocate(ibuffr(pbuffr), stat = error)
      if (error /= 0) then
         ! space for these arrays could not be allocated
         print *, "error: (hertz) could not allocate space."
         stop
      end if  ! (error /= 0)

      call freebsub ( ibuffr, ibuffr, pbuffr )
      end

!     --- Function subroutine for the boundary conditions

      function funcbc( ichoice, x, y, z)
      implicit none

      integer ichoice
      double precision funcbc, x, y, z

      if (ichoice==1) then

!     --- ichoice = 1, boundary condition for the contact points
!         The z-displacement is made equal to -z
!         In this way points are moved back to z=0

        funcbc = -z

      end if

      end

!     --- Subroutine funcvect defines the contact distance and the
!         contact force

      subroutine funcvect ( ichoice, ndim, coor, numnodes,
     +                      uold, nuold, result, nphys )
      implicit none
```

```
      integer ichoice, ndim, numnodes, nuold, nphys
      double precision coor(ndim, numnodes),
     +                 uold( numnodes, nphys, nuold),
     +                 result( numnodes, *)

      integer k

      if ( ichoice==3 ) then

!     --- ichoice = 3, contact distance = u_z + z

         do k = 1, numnodes
            result(k,1) = coor(3,k) + uold(k,3,1)
         end do

      else if ( ichoice==4 ) then

!     --- ichoice = 4, contact force is third component of reaction force

         do k = 1, numnodes
            result(k,1) = uold(k,3,1)
         end do

      end if

      end
```

Postprocessing may be performed for example by program seppost using the following input file:

```
#  hertz.pst
#
#  Input file for postprocessing for Hertz-problem:
#  See Manual Examples Section 5.5.1
#
#
#  To run this file use:
#     seppost hertz.pst > hertz.out
#
#  Reads the files meshoutput and sepcomp.out
#
#
postprocessing                    # See Users Manual Section 5.2

#  Plot the results
#  See Users Manual Section 5.4
    plot identification, text = 'Hertz contact (cylinder)', origin = (3,18)
    plot boundary function displacement, degfd 3, curves (c6)
    plot boundary function reaction_force, degfd 3, curves (c6)
    plot boundary function contact_distance, curves (c6)
    plot boundary function contact_force, curves (c6)
    plot boundary function stress, degfd 3, curves (c6)
    plot boundary function strain, degfd 3, curves (c6)
end
```

The Hertz example can be made more efficient by changing the loop in the contact problem. Instead of checking on the difference of the contact distance in two succeeding iterations, the process is stopped as soon as the contact region is not changed anymore. This reduces the number of iterations considerably.
The test is performed in a while loop, where common block ccontact is used to see if the contact region is changed or not. This actual check is done in the user function userbool.
This updated example is called hertz2.
To get this example into your local directory use:

```
sepgetex hertz2
```

To run the problem use

```
seplink hertz2
hertz < hertz2.prb
seppost hertz2.pst
```

The files that are (slightly) different from the hertz example are the fortran file `hertz.f` and the input file `hertz2.prb`.
These files are given below

```
      program hertz2

!      --- Main program for the Hertz-problem:
!          An infinitely long, elastic, half cylinder is pressed
!          on a flat surface.
!          A slice of this cylinder is analyzed.
!          This main program is necessary because of the variable boundary
!          conditions

      implicit none

      integer, allocatable, dimension (:) :: ibuffr
      integer pbuffr, error
      parameter ( pbuffr=25000000)
      allocate(ibuffr(pbuffr), stat = error)
      if (error /= 0) then
         ! space for these arrays could not be allocated
         print *, "error: (hertz2) could not allocate space."
         stop
      end if  ! (error /= 0)

      call freebsub ( ibuffr, ibuffr, pbuffr )
      end

!     --- Function subroutine for the boundary conditions

      function funcbc( ichoice, x, y, z)
      implicit none

      integer ichoice
      double precision funcbc, x, y, z

      if (ichoice==1) then
```

```
!     --- ichoice = 1, boundary condition for the contact points
!         The z-displacement is made equal to -z
!         In this way points are moved back to z=0

        funcbc = -z

      end if

      end

!     --- Subroutine funcvect defines the contact distance and the
!         contact force

      subroutine funcvect ( ichoice, ndim, coor, numnodes,
     +                      uold, nuold, result, nphys )
      implicit none

      integer ichoice, ndim, numnodes, nuold, nphys
      double precision coor(ndim, numnodes),
     +                 uold( numnodes, nphys, nuold),
     +                 result( numnodes, *)

      integer k

      if ( ichoice==3 ) then

!     --- ichoice = 3, contact distance = u_z + z

         do k = 1, numnodes
            result(k,1) = coor(3,k) + uold(k,3,1)
         end do

      else if ( ichoice==4 ) then

!     --- ichoice = 4, contact force is third component of reaction force

         do k = 1, numnodes
            result(k,1) = uold(k,3,1)
         end do

      end if

      end

!     --- Function user bool is used to set the boolean
!         In this case the boolean is true if the contact region has been
!         changed

      function userbool( ichoice )
      implicit none
      logical userbool
      integer ichoice

      include 'SPcommon/ccontact'
```

```
      if ( ichoice==1 ) then

!     --- ichoice = 1, the only possible value in this program
!         set userbool equal to the value of contact_changed(1)
!         This indicates if the contact region corresponding to the first
!         (and in this case only) contact problem has been changed

         userbool = contact_changed(1)

      end if

      end

#  hertz2.prb
#
#  Hertz-problem:
#   An infinitely long, elastic, half cylinder is pressed
#     on a flat surface.
#   A slice of this cylinder is analyzed.
#  See Manual Examples Section 5.5.1
#
#  This example is completely identical to hertz
#  The only difference is that the process is stopped as soon as the contact
#  surface remains unchanged
#
#  To run this file use:
#     seplink hertz2
#     hertz2 < hertz2.prb
#
#  Creates the files meshoutput and sepcomp.out
#
#  Define some general constants
#
constants
   vector_names   # names of vectors to be used in the computation
      displacement         # displacement_vector
      reaction_force       # vector with reaction_forces
      contact_distance     # vector in which the contact distance is stored
      contact_force        # vector in which the contact force is stored
      stress               # stress tensor
      strain               # strain tensor
end
#
#  Some information at the start of the program
#
start             # See Users Manual Section 3.2.1
   norotate           # Plots may not be rotated
end
#
#  First we define the mesh in the slice
#
mesh3d             # See Users Manual Section 2.2
  coarse (unit = 0.1)               # define the unit length of elements
                                    # In the contact region at the bottom the
                                    # mesh is refined
```

```
#
#  user points
#
   points           # See Users Manual Section 2.2
    p1 = ( 0.0, 0.0, 1.0, 1.00)     # centre point at front side of top of
                                    # cylinder
    p2 = ( 1.0, 0.0, 1.0, 1.00)     # right-hand side point at front side of top
                                    # of cylinder
    p3 = (-1.0, 0.0, 1.0, 1.00)     # left-hand side point at front side of top
                                    # of cylinder
    p4 = ( 0.0, 0.0, 0.0, 0.25)     # bottom point of front side of cylinder

    p5 = ( 0.0, 0.1, 1.0, 1.00)     # centre point at back side of top of
                                    # cylinder
    p6 = ( 1.0, 0.1, 1.0, 1.00)     # right-hand side point at back side of top
                                    # of cylinder
    p7 = (-1.0, 0.1, 1.0, 1.00)     # left-hand side point at back side of top
                                    # of cylinder
    p8 = ( 0.0, 0.1, 0.0, 0.25)     # bottom point of back side of cylinder
#
#  curves
#
   curves           # See Users Manual Section 2.3
    c1 = cline 1 (p3, p1)           # Line  at front side of top of
                                    # cylinder from left to centre
    c2 = cline 1 (p1, p2)           # Line  at front side of top of
                                    # cylinder from centre to right
    c3 =  carc 1 (p2, p4, p1)       # Right-hand side part of curved part of
                                    # front side of cylinder
    c4 =  carc 1 (p4, p3, p1)       # Left-hand side part of curved part of
                                    # front side of cylinder
    c5 = curves (c1, c2)            # Top of half cylinder (front side)
    c6 = curves (c3, c4)            # Curved part of half cylinder (back side)
    c7 = translate c5 (p7, p5, p6)  # Top of half cylinder (back side)
    c8 = translate c6 (p6, p8, p7)  # Curved part of half cylinder (front side)
    c9 = line 1 (p3, p7, nelm = 1)  # Line from front side to back side the
                                    # left
    c10 = line 1 (p2, p6, nelm = 1) # Line from front side to back side the
                                    # right
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
    s1 = general 5 (c5, c6)              # front end of half cylinder
    s2 = translate s1 (c7, c8)           # back end of half cylinder
    s3 = pipesurface 5 (c5, c7, c9, c10)  # top of half cylinder
    s4 = pipesurface 5 (c6, c8, c10, c9)  # curved envelope of half cylinder
    s5 = ordered surface ((s3,s4))       # total envelope of half cylinder
#
#  volumes
#
  volumes         # See Users Manual Section 2.5
    v1 = pipe 13 (s1, s2, s5)            # Complete half cylinder

  plot, eyepoint = (2.0, -3.0, 2.0)      # make a plot of the mesh
```

```
                                        # See Users Manual Section 2.2
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                          # Define types of elements,
                                  # See Users Manual Section 3.2.2
      elgrp 1 = (type=250)        # Type number for linear elasticity
                                  # See Standard problems Section 5.1
   essbouncond                    # Define where essential boundary conditions are
                                  # given (not the value)
                                  # See Users Manual Section 3.2.2
      degfd 2, surfaces (s1)      # No displacement in y-direction of front end
      degfd 2, surfaces (s2)      # No displacement in y-direction of back end
      surfaces (s3)               # Prescribed displacement in top of half
                                  # cylinder
      degfd 3, contact 1          # The z-displacement is 0 in contact points
end
#
#   Input for the contact algorithm
#
contact, sequence_number = 1     # See Users Manual Section 3.2.16
  contact_surface = s4                      # surface that makes contact
  contact_distance = contact_distance     # vector to be used to store the
                                            # contact distance
  contact_force = contact_force           # vector to be used to store the
                                            # contact force
  contact_method = NEG_DISTANCE             # defines when a point is supposed
                                            # to make contact (in this case
                                            # if the contact distance < 0)
  contact_disable_method = CONTACT_FORCE    # defines when a point is supposed
                                            # to lose contact (in this case
                                            # if the contact force < 0)
end
#
#  Define non-zero essential boundary conditions
#  See Users Manual Section 3.2.5
#
essential boundary conditions, sequence_number = 1
  degfd 3, surfaces (s3), value = -0.2      # The displacement in z-direction of
                                            # the top surface = -0.2
  degfd 3, contact 1, func = 1          # In those points where we have contact
                                        # the displacement is made equal to -z,
                                        # so that the points are moved back to
                                        # z = 0
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because we have a free boundary problem

structure                  # See Users Manual Section 3.2.3
```

```
  write_mesh     # First the mesh is written to the file meshoutput
                 # in order to be used for postprocessing

  # Next create 4 vectors that are used during the analysis
  # The displacement vector and the reaction force vector are set equal to 0
  # They contain 3 degrees of freedom per point
  create_vector, sequence_number = 1, displacement
  create_vector, sequence_number = 1, reaction_force

  # In order to solve the (non-linear) contact problem we define a
  # loop by while ( boolean_expr(1)) ... end_while
  # The loop is finished if the contact region does not change anymore
  # The check iscarried out in subroutine userbool

  while ( boolean_expr(1)) do

   # Store the essential boundary conditions in the displacement vector
   # Since they depend on the contact surface they may change in each step
    prescribe_boundary_conditions, sequence_number = 1, vector = 1
    solve_linear_system, //
        seq_solve = 1, seq_coef = 1, vector = 1, reaction_force = reaction_force

  # The vectors contact_distance and contact_force contain one degree of
  # freedom per unknown
    create_vector, sequence_number = 3, contact_distance
    create_vector, sequence_number = 4, contact_force

   # Compute the contact surface using the input for the contact problem
    compute_contact_surface, sequence_number = 1

  end_while


  # Finally compute the stress and the strain tensors
  derivatives, seq_deriv = 1, seq_coef = 1, stress
  derivatives, seq_deriv = 2, seq_coef = 1, strain

  output
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
  storage_scheme = compact, symmetric, reaction_force
                   # symmetrical matrix with compact storage
                   # hence an iterative linear solver is used
                   # reaction forces must be computed
end

# Input for the linear solver
#  See Users Manual Section 3.2.8

solve, sequence_number = 1
  iteration_method = cg, //
```

```
      start=old_solution, //
      preconditioning=ilu, //
      accuracy = 0.01
end

#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 5.1

coefficients, sequence_number = 1
  elgrp 1 (nparm=45)             # The coefficients are defined by 45 parameters
    icoef 2 = 0                  # type of stress-strain relation
                                 # 0: plane stress
     coef 6 = 1.0                # Elasticity modulus
     coef 7 = 0.3                # Poisson ratio
end

# Create start vectors
#  See Users Manual Section 3.2.10
#  First displacement and reaction force
#  Type solution vector

create vector, sequence_number = 1
  value = 0
end

#  Next contact_distance and contact_force
#  One degree of freedom per point

create vector, sequence_number = 2
  type = vector of special structure v1
  value = 0
end

# Create contact_distance during the iterations
# The contact distance is defined as the sum of the z-displacement and
# the z coordinate
# The summation is carried out in subroutine funcvect

create vector, sequence_number = 3
  type = vector of special structure v1
  surfaces (s4), old_vector = contact_distance, seq_vectors = displacement
end

# Create contact_force during the iterations
# The contact force is equal to the third component of the reaction force
# The extraction is carried out in subroutine funcvect

create vector, sequence_number = 4
  type = vector of special structure v1
  surfaces (s4), old_vector = contact_force, seq_vectors = reaction_force
end

# compute stress
# See Users Manual, Section 3.2.11 and Standard problems Section 5.1
```

```
derivatives, sequence_number = 1
  icheld = 6
end

# compute strain
# See Users Manual, Section 3.2.11 and Standard problems Section 5.1

derivatives, sequence_number = 2
  icheld = 7
end

# write the results to the file sepcomp.out
# See Users Manual, Section 3.2.13

output
end
```

5.5.2 The Roll problem

This problem shows the use of multiple contact blocks. It is comparable to the Hertz example of Section 5.5.1 in that a massive, elastic cylinder with a hole is compressed between two plane, rigid surfaces. In this case contact occurs both on top and on the bottom of the cylinder. The treatment of these contact areas is identical to that for the Hertz example.
To get this example into your local directory use:

```
sepgetex roll
```

To run the problem use

```
seplink roll
roll < roll.prb
seppost roll.pst
```

The outer radius of the cylinder (Ru) is equal to 1 and the inner radius (Ri is equal to 0.6. The centre of the cylinder is taken at $y = 0, z = 0$. In the x-direction the cylinder is supposed to be infinitely long so it is sufficient to take a slice (in this case of thickness 0.1) and to apply symmetry conditions in the x-direction. Also symmetry allows us to use only one half of the cylinder. The top contact surface is defined by $z = Ru - dH$ and the bottom contact surface by $z = -Ru$. In our example dH has the value 0.8, which means that the upper surface and as a consequence the top of the roll is pushed down over a distance of 0.8.

The shape of the slice can be seen easily by the plot of the curves in Figure 5.5.2.1. In this case the

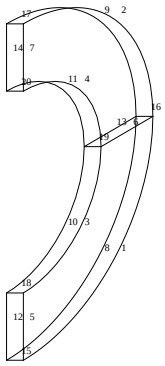


Figure 5.5.2.1: Definition of the curves in the slice

contact distance at the bottom is equal to $u_z + z + Ru$, and at the top equal to $Ru - dH - (u_z + z)$. The mesh and problem file used in this case is:

```
#  roll.prb
#
#  roll-problem:
#   An infinitely long, elastic, hollow cylinder is pressed
#   between two flat surfaces.
#   A slice of this cylinder is analyzed.
#  See Manual Examples Section 5.5.2
#
#  To run this file use:
#     seplink roll
```

```
#     roll < roll.prb
#
#  Creates the files meshoutput and sepcomp.out
#
#  Define some general constants
#
constants
   reals
      Ru = 1.0            # Radius of outer cylinder
      Ri = 0.6            # Radius of inner cylinder
      L = 0.1             # Thickness of cylinder
      dH = 0.8            # Downwards displacement of upper contact surface
   vector_names   # names of vectors to be used in the computation
      displacement         # displacement_vector
      reaction_force       # vector with reaction_forces
      contact_distance     # vector in which the contact distance is stored
      contact_force        # vector in which the contact force is stored
      stress               # stress tensor
      strain               # strain tensor
end
#
#  Some information at the start of the program
#
start             # See Users Manual Section 3.2.1
   norotate           # Plots may not be rotated
end
#
#  First we define the mesh in the slice
#
mesh3d             # See Users Manual Section 2.2
  coarse (unit = 0.1)               # define the unit length of elements
                                    # In the contact region at the bottom and
                                    # the top the mesh is refined
#
#  user points
#
   points           # See Users Manual Section 2.2
    # First points on front side ( x = 0 )
    p1 = ( 0.0, 0.0, 0.0, 1.0)     # centre point of top of cylinder
    #  Outer cylinder
    p2 = ( 0.0, 0.0,- Ru, 0.2)     # point at bottom of outer cylinder
    p3 = ( 0.0,  Ru, 0.0, 1.0)     # point at right-hand side of outer cylinder
    p4 = ( 0.0, 0.0,  Ru, 0.2)     # point at top of outer cylinder
    #  Inner cylinder
    p5 = ( 0.0, 0.0,- Ri, 0.6)     # point at bottom of inner cylinder
    p6 = ( 0.0,  Ri, 0.0, 1.0)     # point at right-hand side of inner cylinder
    p7 = ( 0.0, 0.0,  Ri, 0.6)     # point at top of inner cylinder
    # Next points on back side ( x = -L )
    p8 = ( - L, 0.0, 0.0, 1.0)     # Point opposite to p1
    #  Outer cylinder
    p9 = ( - L, 0.0,- Ru, 0.2)     # Point opposite to p2
    p10= ( - L,  Ru, 0.0, 1.0)     # Point opposite to p3
    p11= ( - L, 0.0,  Ru, 0.2)     # Point opposite to p4
    #  Inner cylinder
    p12= ( - L, 0.0,- Ri, 0.6)     # Point opposite to p5
```

```
    p13= ( - L,  Ri, 0.0, 1.0)     # Point opposite to p6
    p14= ( - L, 0.0,  Ri, 0.6)     # Point opposite to p7
#
#  curves
#
   curves           # See Users Manual Section 2.3
    # First curves on front side ( x = 0 )
    #  Outer cylinder
    c1 = carc ( p2 , p3 , p1 , nodd=2)   # lower part of circle
    c2 = carc ( p3 , p4 , p1 , nodd=2)   # upper part of circle
    #  Inner cylinder
    c3 = carc ( p5 , p6 , p1 , nodd=2)   # lower part of circle
    c4 = carc ( p6 , p7 , p1 , nodd=2)   # upper part of circle
    #  Connection lines between two circles
    c5 = cline ( p5 , p2 , nodd=2)
    c6 = cline ( p6 , p3 , nodd=2)
    c7 = cline ( p7 , p4 , nodd=2)
    # Next curves on back side ( x = -L )
    #  Outer cylinder
    c8 = carc ( p9 , p10, p8 , nodd=2)   # lower part of circle
    c9 = carc ( p10, p11, p8 , nodd=2)   # upper part of circle
    #  Inner cylinder
    c10= carc ( p12, p13, p8 , nodd=2)   # lower part of circle
    c11= carc ( p13, p14, p8 , nodd=2)   # upper part of circle
    #  Connection lines between two circles
    c12= cline ( p12, p9 , nodd=2)
    c13= cline ( p13, p10, nodd=2)
    c14= cline ( p14, p11, nodd=2)
    #  Connection lines between front side and back side
    #  Outer cylinder
    c15= line ( p2 , p9 , nelm=1)
    c16= line ( p3 , p10, nelm=1)
    c17= line ( p4 , p11, nelm=1)
    #  Inner cylinder
    c18= line ( p5 , p12, nelm=1)
    c19= line ( p6 , p13, nelm=1)
    c20= line ( p7 , p14, nelm=1)
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
    # First surfaces on front side ( x = 0 )
    s1 = general 5 ( c1 ,-c6 ,-c3 , c5 )   # lower part of circle
    s2 = general 5 ( c2 ,-c7 ,-c4 , c6 )   # upper part of circle
    # Next surfaces on back side ( x = -L )
    s3 = general 5 ( c8 ,-c13,-c10, c12)   # lower part of circle
    s4 = general 5 ( c9 ,-c14,-c11, c13)   # upper part of circle
    # enveloping surfaces
    s5 = pipesurface 5 ( c5 , c12, c18, c15)
    s6 = pipesurface 5 ( c7 , c14, c20, c17)

    s7 = pipesurface 5 ( c1 , c8 , c15, c16)  # bottom contact surface
    s8 = pipesurface 5 ( c2 , c9 , c16, c17)  # top contact surface

    s9 = pipesurface 5 ( c3 , c10, c18, c19)
```

```
    s10= pipesurface 5 ( c4 , c11, c19, c20)
    # Reorganization into 3 surfaces
    s11= surfaces ( s1 , s2 )  # front side
    s12= surfaces ( s3 , s4 )  # back side
    s13= ordered surfaces (( s7 , s8 ,-s6 ,-s10,-s9 , s5 )) #envelope
#
#  volumes
#
  volumes         # See Users Manual Section 2.5
    v1 = pipe 13 ( s11, s12, s13)

  plot, eyepoint = (2.0, 0.5, 0.5)        # make a plot of the mesh
                                          # See Users Manual Section 2.2
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp 1 = (type=250)      # Type number for linear elasticity
                                # See Standard problems Section 5.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
    degfd 1, surfaces (s3, s4)  # No displacement in y-direction of front end
    degfd 2, surfaces (s5, s6)  # No displacement in y-direction of back end
    degfd 3, contact 1          # The z-displacement in the bottom contact
                                # surface is prescribed
    degfd 3, contact 2          # The z-displacement in the top contact
                                # surface is prescribed
end
#
#   Input for the contact algorithm
#
contact, sequence_number = 1     # See Users Manual Section 3.2.16
  contact_surface = s7           # contact at bottom contact surface
  contact_distance = contact_distance    # vector to be used to store the
                                         # contact distance
  contact_force = contact_force          # vector to be used to store the
                                         # contact force
  contact_method = NEG_DISTANCE          # defines when a point is supposed
                                         # to make contact (in this case
                                         # if the contact distance < 0)
  contact_disable_method = CONTACT_FORCE # defines when a point is supposed
                                         # to lose contact (in this case
                                         # if the contact force < 0)
end

contact, sequence_number = 2
  contact_surface = s8           # contact at top contact surface
  contact_distance = contact_distance    # vector to be used to store the
                                         # contact distance
  contact_force = contact_force          # vector to be used to store the
```

```
                                          # contact force
  contact_method = NEG_DISTANCE           # defines when a point is supposed
                                          # to make contact (in this case
                                          # if the contact distance < 0)
  contact_disable_method = CONTACT_FORCE  # defines when a point is supposed
                                          # to lose contact (in this case
                                          # if the contact force < 0)
end
#
#  Define non-zero essential boundary conditions
#  See Users Manual Section 3.2.5
#
essential boundary conditions, sequence_number = 1
  degfd 3, contact 1, func = 11  # In those points of the bottom contact surface
                                 # where we have contact the displacement is
                                 # made equal to -Ru-z, so that the points are
                                 # moved back to z = -Ru
  degfd 3, contact 2, func = 12  # In those points of the top contact surface
                                 # where we have contact the displacement is
                                 # made equal to Ru-dH-z, so that the points are
                                 # moved back to z = Ru-dH
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because we have a free boundary problem

structure                   # See Users Manual Section 3.2.3

  write_mesh     # First the mesh is written to the file meshoutput
                 # in order to be used for postprocessing

  # Next create 4 vectors that are used during the analysis
  # The displacement vector and the reaction force vector are set equal to 0
  # They contain 3 degrees of freedom per point

  create_vector, sequence_number = 1, displacement
  create_vector, sequence_number = 1, reaction_force

  # The vectors contact_distance and contact_force contain one degree of
  # freedom per unknown and are also initialized to 0
  create_vector, sequence_number = 2, contact_distance
  create_vector, sequence_number = 2, contact_force

  # In order to solve the (non-linear) contact problem we define a
  # loop by start_loop ... end_loop

  start_loop, sequence_number = 1
   # Compute the contact surfaces using the input for the contact problem
   # First the bottom contact surface
    compute_contact_surface, sequence_number = 1
   # Next the top contact surface
    compute_contact_surface, sequence_number = 2

   # Store the essential boundary conditions in the displacement vector
```

```
   # Since they depend on the contact surface they may change in each step
    prescribe_boundary_conditions, sequence_number = 1, displacement

   # Solve the displacement vector by the linear elasticity problem
   # Compute the reaction force vector, necessary for the contact
   # algorithm
    solve_linear_system, //
        seq_solve = 1, seq_coef = 1, displacement//
        reaction_force = reaction_force

   # Recompute the contact distance and the contact force
    create_vector, sequence_number = 3, contact_distance
    create_vector, sequence_number = 4, contact_force
  end_loop

  # Finally compute the stress and the strain tensors
  derivatives, seq_deriv = 1, seq_coef = 1, stress
  derivatives, seq_deriv = 2, seq_coef = 1, strain

  output
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
  storage_scheme = compact, symmetric, reaction_force
                   # symmetrical matrix with compact storage
                   # hence an iterative linear solver is used
                   # reaction forces must be computed
end

#  Input for the loop in the structure block
#  Defines how many iterations may be carried out at most
#  and when the process is finished
#  See Users Manual Section 3.2.3

loop_input, sequence_number = 1
  maxiter = 200                    # maximum number of iterations
  miniter = 2                      # minimum number of iterations
  accuracy = 1d-5                  # relative accuracy
  criterion = relative
  seq_vector = displacement        # vector to be used to check the convergence
end

# Input for the linear solver
#  See Users Manual Section 3.2.8

solve, sequence_number = 1
  iteration_method = cg, //
      start=old_solution, //
      preconditioning=ilu, //
      accuracy = 0.01
end
```

```
#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 5.1

coefficients, sequence_number = 1
  elgrp 1 (nparm=45)            # The coefficients are defined by 45 parameters
    icoef 2 = 0                 # type of stress-strain relation
                                # 0: plane stress
     coef 6 = 1.0               # Elasticity modulus
     coef 7 = 0.3               # Poisson ratio
end

# Create start vectors
#  See Users Manual Section 3.2.10
#  First displacement and reaction force
#  Type solution vector

create vector, sequence_number = 1
  value = 0
end

#  Next contact_distance and contact_force
#  One degree of freedom per point

create vector, sequence_number = 2
  type = vector of special structure v1
  value = 0
end

# Create contact_distance during the iterations
# The contact distance is defined as the sum of the z-displacement and
# the z coordinate
# The summation is carried out in subroutine funcvect

create vector, sequence_number = 3
  type = vector of special structure displacement
  surfaces (s7), old_vector = 31, seq_vectors = displacement
  surfaces (s8), old_vector = 32, seq_vectors = displacement
end

# Create contact_force during the iterations
# The contact force is equal to the third component of the reaction force
# The extraction is carried out in subroutine funcvect

create vector, sequence_number = 4
  type = vector of special structure displacement
  surfaces (s7), old_vector = 41, seq_vectors = reaction_force
  surfaces (s8), old_vector = 42, seq_vectors = reaction_force
end

# compute stress
# See Users Manual, Section 3.2.11 and Standard problems Section 5.1

derivatives, sequence_number = 1
  icheld = 6
```

```
end

# compute strain
# See Users Manual, Section 3.2.11 and Standard problems Section 5.1

derivatives, sequence_number = 2
  icheld = 7
end

# write the results to the file sepcomp.out
# See Users Manual, Section 3.2.13

output
end
```

Figure 5.5.2.2 shows the mesh used in this problem This file requires a main program with subrou-

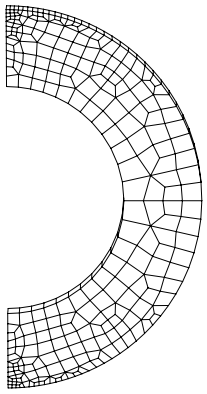


Figure 5.5.2.2: Mesh created in the slice

tines, since the boundary condition in the contact surface depends on space and in order to compute the contact distance and contact force. The main program used by us is:

```
      program roll

!      --- Main program for the Roll-problem:
!          An infinitely long, elastic, cylinder is pressed
!          between two flat surfaces.
!          A slice of this cylinder is analyzed.
!          This main program is necessary because of the variable boundary
!          conditions

      implicit none

      integer, allocatable, dimension (:) :: ibuffr
      integer pbuffr, error
      parameter ( pbuffr=25000000)
      allocate(ibuffr(pbuffr), stat = error)
      if (error /= 0) then
         ! space for these arrays could not be allocated
         print *, "error: (roll) could not allocate space."
```

```
         stop
      end if  ! (error /= 0)

      call freebsub ( ibuffr, ibuffr, pbuffr )
      end

!     --- Function subroutine for the boundary conditions


      function funcbc( ichoice, x, y, z)
      implicit none

      integer ichoice
      double precision funcbc, x, y, z
      integer ifirst
      double precision Ru, dH
      double precision getconst
      save Ru, dH
      data ifirst /0/

      if ( ifirst==0 ) then

!     --- ifirst = 0, first call of funcbc
!         Get the values of some constants

         ifirst = 1
         Ru = getconst ( 'Ru' )
         dH = getconst ( 'dH' )

      end if

      if (ichoice==11) then

!     --- ichoice = 11, bottom contact surface
!         In order to restrict the solution to z = -Ru it is necessary
!         to set the displacement equal to -Ru - z

        funcbc = -Ru - z

      else if (ichoice==12) then

!     --- ichoice = 12, top contact surface
!         In order to restrict the solution to z = Ru-dH it is necessary
!         to set the displacement equal to Ru-dH - z

        funcbc =  Ru-dH - z

      end if

      end

!     --- Subroutine funcvect defines the contact distance and the
!         contact force

      subroutine funcvect ( ichoice, ndim, coor, numnodes,
```

```
     +                     uold, nuold, result, nphys )
      implicit none

      integer ichoice, ndim, numnodes, nuold, nphys
      double precision coor(ndim, numnodes),
     +                 uold( numnodes, nphys, nuold),
     +                 result( numnodes, *)

      integer k, ifirst
      double precision Ru, dH
      save Ru, dH
      double precision getconst
      data ifirst /0/

      if ( ifirst==0 ) then

!     --- ifirst = 0, first call of funcbc
!         Get the values of some constants

         ifirst = 1
         Ru = getconst ( 'Ru' )
         dH = getconst ( 'dH' )

      end if

      if (ichoice==31) then

!     --- ichoice = 31, contact distance = u_z + z + Ru

        do k = 1, numnodes
           result(k,1) = (coor(3,k) + uold(k,3,1)) + Ru
        end do

      else if (ichoice==32) then

!     --- ichoice = 32, contact distance =  + Ru-dH- (u_z + z)

        do k = 1, numnodes
           result(k,1) = Ru-dH - (coor(3,k) + uold(k,3,1))
        end do

      else if (ichoice==41) then

!     --- ichoice = 41, contact force is third component of reaction force

        do k = 1, numnodes
           result(k,1) = uold(k,3,1)
        end do

      else if (ichoice==42) then

!     --- ichoice = 42, contact force is minus third component of reaction force

        do k = 1, numnodes
           result(k,1) = -uold(k,3,1)
```

```
  end do
end if
end
```

The post processing file is almost the same as in the Hertz problem (5.5.1)

5.5.3 The Wheel problem

An elastic layer (the "tire") is fixed to a rigid cylinder (the "hub"). This hub is compressed downwards and the tire is pressed onto the "road". The contact area increases for increasing load. The contact algorithm used here is identical to that for the Hertz example of Section 5.5.1. Furthermore there is a large resemblance with the Roll problem of Section 5.5.2. In fact the definition of the curves is the same and the only difference in the mesh is that since contact is made on the lower boundary only, no refinement in the top is applied.
To get this example into your local directory use:

```
sepgetex wheel
```

To run the problem use

```
seplink wheel
wheel < wheel.prb
seppost wheel.pst
```

The load on the wheel is simulated by prescribing the z-displacement of the hub in downwards direction. This is effectuated by prescribing the displacement in the inner cylinder.
The mesh and problem file used in this case is:

```
#  wheel.prb
#
#  wheel-problem:
#   An elastic layer (the "tire") is fixed to a rigid cylinder (the "hub").
#   This hub is compressed downwards and the tire is pressed onto the
#   "road".  The contact area increases for increasing load.
#  See Manual Examples Section 5.5.3
#
#  To run this file use:
#     seplink wheel
#     wheel < wheel.prb
#
#  Creates the files meshoutput and sepcomp.out
#
#  Define some general constants
#
constants
   reals
      Ru = 1.0            # Radius of outer cylinder
      Ri = 0.6            # Radius of inner cylinder
      L = 0.1             # Thickness of cylinder
      dH = 0.3            # Downwards displacement of upper contact surface
   vector_names   # names of vectors to be used in the computation
      displacement         # displacement_vector
      reaction_force       # vector with reaction_forces
      contact_distance     # vector in which the contact distance is stored
      contact_force        # vector in which the contact force is stored
      stress               # stress tensor
      strain               # strain tensor
end
#
#  Some information at the start of the program
#
```

```
start             # See Users Manual Section 3.2.1
   norotate           # Plots may not be rotated
end
#
#  First we define the mesh in the slice
#  The mesh is almost identical to the mesh in the roll problem
#  except that refinement is only applied at the bottom
#
mesh3d             # See Users Manual Section 2.2
  coarse (unit = 0.1)               # define the unit length of elements
                                    # In the contact region at the bottom and
                                    # the top the mesh is refined
#
#  user points
#
   points           # See Users Manual Section 2.2
    # First points on front side ( x = 0 )
    p1 = ( 0.0, 0.0, 0.0, 1.0)     # centre point of top of cylinder
    #  Outer cylinder
    p2 = ( 0.0, 0.0,- Ru, 0.2)     # point at bottom of outer cylinder
    p3 = ( 0.0,  Ru, 0.0, 1.0)     # point at right-hand side of outer cylinder
    p4 = ( 0.0, 0.0,  Ru, 1.0)     # point at top of outer cylinder
    #  Inner cylinder
    p5 = ( 0.0, 0.0,- Ri, 0.6)     # point at bottom of inner cylinder
    p6 = ( 0.0,  Ri, 0.0, 1.0)     # point at right-hand side of inner cylinder
    p7 = ( 0.0, 0.0,  Ri, 1.0)     # point at top of inner cylinder
    # Next points on back side ( x = -L )
    p8 = ( - L, 0.0, 0.0, 1.0)     # Point opposite to p1
    #  Outer cylinder
    p9 = ( - L, 0.0,- Ru, 0.2)     # Point opposite to p2
    p10= ( - L,  Ru, 0.0, 1.0)     # Point opposite to p3
    p11= ( - L, 0.0,  Ru, 1.0)     # Point opposite to p4
    #  Inner cylinder
    p12= ( - L, 0.0,- Ri, 0.6)     # Point opposite to p5
    p13= ( - L,  Ri, 0.0, 1.0)     # Point opposite to p6
    p14= ( - L, 0.0,  Ri, 1.0)     # Point opposite to p7
#
#  curves
#
   curves           # See Users Manual Section 2.3
    # First curves on front side ( x = 0 )
    #  Outer cylinder
    c1 = carc ( p2 , p3 , p1 , nodd=2)   # lower part of circle
    c2 = carc ( p3 , p4 , p1 , nodd=2)   # upper part of circle
    #  Inner cylinder
    c3 = carc ( p5 , p6 , p1 , nodd=2)   # lower part of circle
    c4 = carc ( p6 , p7 , p1 , nodd=2)   # upper part of circle
    #  Connection lines between two circles
    c5 = cline ( p5 , p2 , nodd=2)
    c6 = cline ( p6 , p3 , nodd=2)
    c7 = cline ( p7 , p4 , nodd=2)
    # Next curves on back side ( x = -L )
    #  Outer cylinder
    c8 = carc ( p9 , p10, p8 , nodd=2)   # lower part of circle
    c9 = carc ( p10, p11, p8 , nodd=2)   # upper part of circle
```

```
    #  Inner cylinder
    c10= carc ( p12, p13, p8 , nodd=2)   # lower part of circle
    c11= carc ( p13, p14, p8 , nodd=2)   # upper part of circle
    #  Connection lines between two circles
    c12= cline ( p12, p9 , nodd=2)
    c13= cline ( p13, p10, nodd=2)
    c14= cline ( p14, p11, nodd=2)
    #  Connection lines between front side and back side
    #  Outer cylinder
    c15= line ( p2 , p9 , nelm=1)
    c16= line ( p3 , p10, nelm=1)
    c17= line ( p4 , p11, nelm=1)
    #  Inner cylinder
    c18= line ( p5 , p12, nelm=1)
    c19= line ( p6 , p13, nelm=1)
    c20= line ( p7 , p14, nelm=1)
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
    # First surfaces on front side ( x = 0 )
    s1 = general 5 ( c1 ,-c6 ,-c3 , c5 )   # lower part of circle
    s2 = general 5 ( c2 ,-c7 ,-c4 , c6 )   # upper part of circle
    # Next surfaces on back side ( x = -L )
    s3 = general 5 ( c8 ,-c13,-c10, c12)   # lower part of circle
    s4 = general 5 ( c9 ,-c14,-c11, c13)   # upper part of circle
    # enveloping surfaces
    s5 = pipesurface 5 ( c5 , c12, c18, c15)
    s6 = pipesurface 5 ( c7 , c14, c20, c17)

    s7 = pipesurface 5 ( c1 , c8 , c15, c16)  # contact surface
    s8 = pipesurface 5 ( c2 , c9 , c16, c17)  # top surface

    s9 = pipesurface 5 ( c3 , c10, c18, c19)
    s10= pipesurface 5 ( c4 , c11, c19, c20)
    # Reorganization into 3 surfaces
    s11= surfaces ( s1 , s2 )  # front side
    s12= surfaces ( s3 , s4 )  # back side
    s13= ordered surfaces (( s7 , s8 ,-s6 ,-s10,-s9 , s5 )) #envelope
#
#  volumes
#
  volumes         # See Users Manual Section 2.5
    v1 = pipe 13 ( s11, s12, s13)

  plot, eyepoint = (2.0, 0.5, 0.5)        # make a plot of the mesh
                                          # See Users Manual Section 2.2
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
```

```
      elgrp 1 = (type=250)      # Type number for linear elasticity
                                # See Standard problems Section 5.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
    degfd 1, surfaces (s3, s4)  # No displacement in y-direction of front end
    degfd 2, surfaces (s5, s6)  # No displacement in y-direction of back end
    surfaces (s9, s10)          # No displacement at inner side of the wheel
    degfd 3, contact 1          # The z-displacement in the contact
                                # surface is prescribed
end
#
#   Input for the contact algorithm
#
contact, sequence_number = 1     # See Users Manual Section 3.2.16
  contact_surface = s7           # contact at contact surface
  contact_distance = contact_distance    # vector to be used to store the
                                           # contact distance
  contact_force = contact_force          # vector to be used to store the
                                           # contact force
  contact_method = NEG_DISTANCE            # defines when a point is supposed
                                           # to make contact (in this case
                                           # if the contact distance < 0)
  contact_disable_method = CONTACT_FORCE   # defines when a point is supposed
                                           # to lose contact (in this case
                                           # if the contact force < 0)
end
#
#  Define non-zero essential boundary conditions
#  See Users Manual Section 3.2.5
#
essential boundary conditions, sequence_number = 1
  degfd 3, contact 1, func = 11  # In those points of the contact surface
                                 # where we have contact the displacement is
                                 # made equal to -Ru-z, so that the points are
                                 # moved back to z = -Ru
  degfd 3, surfaces (s9, s10), value = - dH  # The load on the wheel is
                                 # represented by a vertical (downwards)
                                 # displacement of the inner side of the wheel
                                 # 9the hub)
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because we have a free boundary problem

structure                   # See Users Manual Section 3.2.3

  write_mesh     # First the mesh is written to the file meshoutput
                 # in order to be used for postprocessing

  # Next create 4 vectors that are used during the analysis
  # The displacement vector and the reaction force vector are set equal to 0
  # They contain 3 degrees of freedom per point
```

```
  create_vector, sequence_number = 1, displacement
  create_vector, sequence_number = 1, reaction_force

  # The vectors contact_distance and contact_force contain one degree of
  # freedom per unknown and are also initialized to 0
  create_vector, sequence_number = 2, contact_distance
  create_vector, sequence_number = 2, contact_force

  # In order to solve the (non-linear) contact problem we define a
  # loop by start_loop ... end_loop

  start_loop, sequence_number = 1
   # Compute the contact surfaces using the input for the contact problem
    compute_contact_surface, sequence_number = 1

   # Store the essential boundary conditions in the displacement vector
   # Since they depend on the contact surface they may change in each step
    prescribe_boundary_conditions, sequence_number = 1, displacement

   # Solve the displacement vector by the linear elasticity problem
   # Compute the reaction force vector, necessary for the contact
   # algorithm
    solve_linear_system, //
        seq_solve = 1, seq_coef = 1, displacement//
        reaction_force = reaction_force

   # Recompute the contact distance and the contact force
    create_vector, sequence_number = 3, contact_distance
    create_vector, sequence_number = 4, contact_force
  end_loop

  # Finally compute the stress and the strain tensors
  derivatives, seq_deriv = 1, seq_coef = 1, stress
  derivatives, seq_deriv = 2, seq_coef = 1, strain

  output
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
  storage_scheme = compact, symmetric, reaction_force
                   # symmetrical matrix with compact storage
                   # hence an iterative linear solver is used
                   # reaction forces must be computed
end

#  Input for the loop in the structure block
#  Defines how many iterations may be carried out at most
#  and when the process is finished
#  See Users Manual Section 3.2.3

loop_input, sequence_number = 1
  maxiter = 200                    # maximum number of iterations
```

```
  miniter = 2                      # minimum number of iterations
  accuracy = 1d-5                  # relative accuracy
  criterion = relative
  seq_vector = displacement        # vector to be used to check the convergence
end

# Input for the linear solver
#  See Users Manual Section 3.2.8

solve, sequence_number = 1
  iteration_method = cg, //
      start=old_solution, //
      preconditioning=ilu, //
      accuracy = 0.01
end

#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 5.1

coefficients, sequence_number = 1
  elgrp 1 (nparm=45)             # The coefficients are defined by 45 parameters
    icoef 2 = 0                  # type of stress-strain relation
                                 # 0: plane stress
     coef 6 = 1.0                # Elasticity modulus
     coef 7 = 0.3                # Poisson ratio
end

# Create start vectors
#  See Users Manual Section 3.2.10
#  First displacement and reaction force
#  Type solution vector

create vector, sequence_number = 1
  value = 0
end

#  Next contact_distance and contact_force
#  One degree of freedom per point

create vector, sequence_number = 2
  type = vector of special structure v1
  value = 0
end

# Create contact_distance during the iterations
# The contact distance is defined as the sum of the z-displacement and
# the z coordinate
# The summation is carried out in subroutine funcvect

create vector, sequence_number = 3
  type = vector of special structure displacement
  surfaces (s7), old_vector = 31, seq_vectors = displacement
end
```

```
# Create contact_force during the iterations
# The contact force is equal to the third component of the reaction force
# The extraction is carried out in subroutine funcvect

create vector, sequence_number = 4
  type = vector of special structure displacement
  surfaces (s7), old_vector = 41, seq_vectors = reaction_force
end

# compute stress
# See Users Manual, Section 3.2.11 and Standard problems Section 5.1

derivatives, sequence_number = 1
  icheld = 6
end

# compute strain
# See Users Manual, Section 3.2.11 and Standard problems Section 5.1

derivatives, sequence_number = 2
  icheld = 7
end

# write the results to the file sepcomp.out
# See Users Manual, Section 3.2.13

output
end
```

The main program used by us is:

```
      program wheel

!      --- Main program for the Wheel-problem:
!          An elastic layer (the "tire") is fixed to a rigid cylinder
!          (the "hub").
!          This hub is compressed downwards and the tire is pressed onto the
!         "road".  The contact area increases for increasing load.
!          This main program is necessary because of the variable boundary
!          conditions

      implicit none

      integer, allocatable, dimension (:) :: ibuffr
      integer pbuffr, error
      parameter ( pbuffr=25000000)
      allocate(ibuffr(pbuffr), stat = error)
      if (error /= 0) then
         ! space for these arrays could not be allocated
         print *, "error: (wheel) could not allocate space."
         stop
      end if  ! (error /= 0)

      call freebsub ( ibuffr, ibuffr, pbuffr )
      end
```

```
!     --- Function subroutine for the boundary conditions

      function funcbc( ichoice, x, y, z)
      implicit none

      integer ichoice
      double precision funcbc, x, y, z

      integer ifirst
      double precision Ru
      double precision getconst
      save Ru
      data ifirst /0/

      if ( ifirst==0 ) then

!     --- ifirst = 0, first call of funcbc
!         Get the values of some constants

         ifirst = 1
         Ru = getconst ( 'Ru' )

      end if

      if (ichoice==11) then

!     --- ichoice = 11, contact surface
!         In order to restrict the solution to z = -Ru it is necessary
!         to set the displacement equal to -Ru - z

        funcbc =  -Ru - z

      end if

      end

      subroutine funcvect ( ichoice, ndim, coor, numnodes,
     +                      uold, nuold, result, nphys )
      implicit none

      integer ichoice, ndim, numnodes, nuold, nphys
      double precision coor(ndim, numnodes),
     +                 uold( numnodes, nphys, nuold),
     +                 result( numnodes, *)

      integer k, ifirst
      double precision Ru
      save Ru
      double precision getconst
      data ifirst /0/

      if ( ifirst==0 ) then

!     --- ifirst = 0, first call of funcbc
```

```
!         Get the values of some constants

          ifirst = 1
          Ru = getconst ( 'Ru' )

       end if

       if (ichoice==31) then

!      --- ichoice = 31, contact distance = u_z + z + Ru

         do k = 1, numnodes
            result(k,1) = (coor(3,k) + uold(k,3,1)) + Ru
         end do

       else if (ichoice==41) then

!      --- ichoice = 41, contact force is third component of reaction force

         do k = 1, numnodes
            result(k,1) = uold(k,3,1)
         end do

       end if

       end
```

The post processing file is almost the same as in the Hertz problem (5.5.1)

6 Solidification problems

6.1 A fixed grid method: the enthalpy method

6.1.1 Enthalpy approach by non-linear over-relaxation

6.1.1.1 A classical semi-infinite half-space solidification problem

In this example we consider a classical Stefan problem for which an analytic solution is available, Chun and Park (2000). In this example, which is essentially one dimensional, we consider a semi-infinite half space. We start with a liquid with constant temperature. On the left-hand side a constant temperature below the melting temperature is imposed. So the liquid starts freezing. We solve this problem on a one-dimensional mesh and also as illustration on a two-dimensional one. For the 1D case we consider two sets of parameters.
To get these examples into your local directory use:

```
sepgetex enthalpyxd_y
```

with x and y one-digit numbers.
and to run it use:

```
sepmesh enthalpyxd_y.msh
sepcomp enthalpyxd_y.prb
seppost enthalpyxd_y.pst
```

After the first and last step you may view the results using sepview.

The following values for `x` are available:

```
x = 1, 2
```

and for `y`:

```
y = 1 to 2
```

Not all combinations of `x` and `y` have been programmed yet.
x defines the dimension of the space and y the sequence number of the parameter set.

1D Stefan problem with equal parameters for both phases

This is the most simple 1D case in which the parameters for liquid and solid phase are the same. Following Chun and Park (2000) we use the following set:

ρ	1 kg/m^3
κ	2 W/m °C
c_p	2.5×10^6 J/kg°C
L	10^8 J/kg

The initial temperature is set to $2°C$, the melting temperature $T_m = 0°C$.
On the left-hand side a Dirichlet boundary condition is given: T = $-4°C$.
On the right-hand side, the region is cut at $x = 10m$, and since this is an infinite half space, the solution may not change on that boundary. So the natural boundary condition $\kappa \partial T/\partial n = 0$ is used, which implies that no action is required in the FEM formulation. The computation is carried out for 30 days, with a step size Δx of $0.1m$.
The mesh is defined by the following mesh input file

```
#  enthalpy1d_1.msh
#
#  mesh file for 1d stefan problem with equal parameters in both phases
#  The enthalpy method is applied
#  Solution by over-relaxation
#  See Manual Examples Section 6.1.1.1
#
#  To run this file use:
#     sepmesh enthalpy1d_1.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      length = 10      # length of the region in meters
   integers
      n = 100          # number of elements
      lin = 1          # linear elements
end
#
#  Define the mesh
#
mesh1d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1 = 0              # Left point
      p2 = $length        # Right point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Linear elements are used
      c1=line $lin (p1,p2,nelm=$n)     # lower boundary

   plot, nodes = 1               # make a plot of the mesh and plot all nodes
                                 # See Users Manual Section 2.2
```

```
end
```

For an explanation of the input file for sepcomp see the manual Standard Problems Section 6.1.1.

```
# enthalpy1d_1.prb
#
#  problem file for 1d stefan problem with equal parameters in both phases
#  The enthalpy method is applied
#  Solution by over-relaxation
#  See Manual Examples Section 6.1.1.1
#
#  To run this file use:
#     sepcomp enthalpy1d_1.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      kappa = 2                 # thermal conductivity (kg/m^3)
      rho   = 1                 # density  (kg/m^3)
      t0    = 0                 # initial time
      hour  = 3600              # number of seconds in an hour
      day   = {$hour*24}        # number of seconds in a day
      dt    = {6*$hour}         # time step (6h)
      t_end = {30*$day}         # end time (30 days)
      kappa_s = $kappa          # thermal conductivity (solid)
      kappa_l = $kappa          # thermal conductivity (liquid)
      latent_heat = 1e8         # Latent heat (J/kg)
      capacity_s = 2.5d6        # specific heat (solid) (J/kg degree C)
      capacity_l = 2.5d6        # specific heat (liquid)
      melt_temp = 0             # melting temperature (degree C)
   vector_names
      Temperature               # temperature vector
      Enthalpy                  # enthalpy vector
end
#
#  Define the type of problem to be solved
#
problem                  # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=800               # Type number for second order elliptic equation
                               # See Standard problems Section 3.1
                               # Is also used to solve the heat equation
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      points p1                # left-hand side point
end
```

```
#
#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4
#
matrix
   method = 9       # compact matrix, stored per row
                    # necessary for overrelaxation
end

# Define the initial temperature
#  See Users Manual Section 3.2.10

create vector
   value = 2                     # initial Temperature (degree C)
end

# Define the essential boundary conditions
#  See Users Manual Section 3.2.5

essential boundary conditions
   points, p1, value=-4         # boundary Temperature (degree C)
end

#  Define the coefficients for heat equation
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients
   elgrp1(nparm=20)
      coef6  = 1                 # kappa in Kirchoff Temperature,
                                 # must be 1, see Standard Problems 6.2
      coef17 = 1                 # rho*c in enthalpy,
                                 # must be 1, see Standard Problems 6.2
end

#  Input for time integration
#  See Users Manual Section 3.2.15

time_integration
   method = euler_implicit       # Integration by Euler implicit
   tinit = $t0                   # initial time
   tend  = $t_end                # end time
   tstep = $dt                   # time step
   toutinit = $t0                # initial time for output
   toutend  = $t_end             # end time for output
   toutstep = $dt                # time step for output
   seq_coefficients = 1          # sequence number for coefficients (default)
   seq_solution_method = 1       # sequence number for linear solver (default)
   mass_matrix = constant        # mass matrix is constant for each time
   stiffness_matrix = constant   # stiffness matrix is constant for each time
   right_hand_side = zero        # no source
end

#  Input for enthalpy integration
#  See Manual Standard Problems Section 6.1.1
```

```
enthalpy_integration
  seq_time_integration = 1       # refers to time integration input (default)
  seq_boundary_conditions = 1    # refers to essential boundary conditions
                                 # default
                                 # All other parameters are given in the block
                                 # constants
end

#  Define which linear solver must be used and what accuracy is required
#  Overrelaxation is used
#  See Users Manual Section 3.2.8

solve
   iteration_method = overrelaxation, omega = 1, max_iter = 1000//
   niter1 = 5, niter2 = 10, print_level= 0   # omega must be reset each time
                              # step
                              # niter1 and niter2 are used to estimate a
                              # value for omega in each step
                              # These values do not have to be optimal
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved

structure
  # Fill initial condition for the temperature
   create_vector, vector %Temperature
  # Compute the initial enthalpy
   compute_enthalpy
  # Write both vectors to sepcomp.out
   output, sequence_number=1

  # Time loop
   start_time_loop
     # Raise time and compute new temperature and enthalpy
      enthalpy_integration
     # Write both vectors to sepcomp.out
      output, sequence_number=1
   end_time_loop
end
```

Finally the postprocessing file has the following contents

```
# enthalpy1d_1.pst
#
#  Input file for postprocessing for 1d stefan problem with equal parameters
#  in both phases
#  The enthalpy method is applied
#  Solution by over-relaxation
#  See Manual Examples Section 6.1.1.1
#
#  To run this file use:
#     seppost enthalpy1d_1.pst > enthalpy1d_1.out
#
```

```
#  Reads the files meshoutput and sepcomp.out
#
#  Define some general constants
#
constants
   reals
      day = {1/(3600*24)}    # 1/ (number of seconds in a day)
end
#
postprocessing                    # See Users Manual Section 5.2

   time = (0, 2592000, 10)   # Do for each ten-th time step
      plot function V%Temperature, one_picture//
         texty ='Temperature', noplot_legenda
      plot function V%Enthalpy, one_picture, factor=1d-8//
         texty ='Enthalpy (*1e8)', noplot_legenda

#  Plot the time history of enthalpy and temperature at = 0.3
#  The time scale is made in days, so we have to divide the x scale
#  by the number of seconds in a day
#  The enthalpy is scaled by a factor of 10^-8

  time history plot point (0.3) V%Enthalpy, xscale=$day, factor=1d-8//
     textx = 'time (days)', texty = 'Enthalpy at x=0.3 (*1e8)', noplot_legenda
  time history plot point (0.3) V%Temperature, xscale=$day//
     textx = 'time (days)', texty ='Temperature at x=0.3', noplot_legenda

end
```

Figure 6.1.1.1 shows the temperature plotted each tenth step. In Figure 6.1.1.2 the enthalpy (scaled by 10^{-8}) is shown and Figures 6.1.1.3 and 6.1.1.4 contain the time history of the enthalpy and temperature respectively at position x = 0.3.

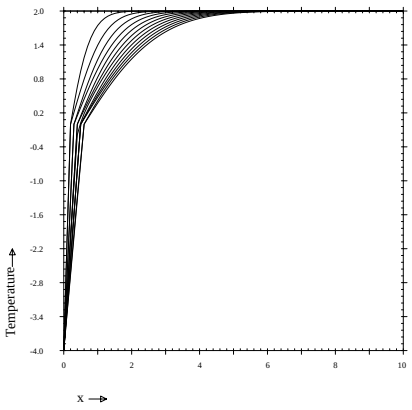


Figure 6.1.1.1: Temperature

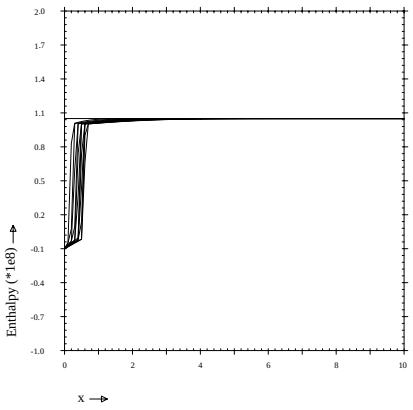


Figure 6.1.1.2: Enthalpy

The *staircase* shape of the temperature at x= 0.3 is inherent to the enthalpy method. It can only be reduced by either using smaller space steps, or refining near the interface.

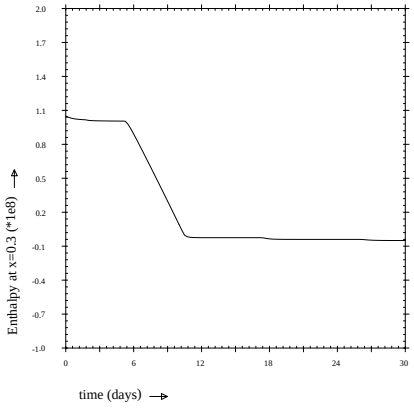


Figure 6.1.1.3: Enthalpy at x=0.3

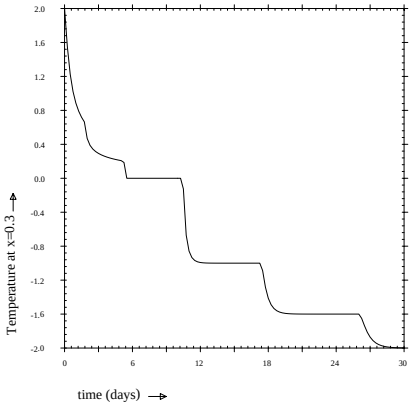


Figure 6.1.1.4: Temperature at x=0.3

6.1.1.2 1D Stefan problem with different parameters for both phases

This is the same example as in Section 6.1.1.1, however with different parameters in solid nd liquid phase. The following set of parameters is used:

$\rho = \rho_s = \rho_l$	1 kg/m^3
κ_s	2.22 W/m °C
κ_l	0.556 W/m °C
cp_s	1.762×10^6 J/kg°C
cp_l	4.226×10^6 J/kg°C
L	3.38×10^8 J/kg

The initial temperature is set to $10°C$, while the melting temperature is again set to be $T_m = 0°C$.
The temperature at the left boundary is kept at $-20°C$.
The time step used is $\Delta t = 2000s$.
The mesh is exactly the same as in Section 6.1.1.1
The problem file now reads

```
# enthalpy1d_2.prb
#
#  problem file for 1d stefan problem with different parameters in both phases
#  The enthalpy method is applied
#  See Manual Examples Section 6.1.1.2
#
#  To run this file use:
#     sepcomp enthalpy1d_2.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      rho    = 1                 # density  (kg/m^3)
      t0     = 0                 # initial time
      hour   = 3600              # number of seconds in an hour
      day    = {24*$hour}        # number of seconds in a day
      dt     = 2000              # time step (seconds)
      t_end  = {30*$day}         # end time (30 days)
      kappa_s = 2.22             # thermal conductivity (solid)
      kappa_l = 0.556            # thermal conductivity (liquid)
      latent_heat = 3.38e8       # Latent heat (J/kg)
      capacity_s = 1.762e6       # specific heat (solid) (J/kg degree C)
      capacity_l = 4.226e6       # specific heat (liquid)
      melt_temp = 0              # melting temperature (degree C)
   vector_names
      Temperature                # temperature vector
      Enthalpy                   # enthalpy vector
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
```

```
                              # See Users Manual Section 3.2.2
      elgrp1=800              # Type number for second order elliptic equation
                              # See Standard problems Section 3.1
                              # Is also used to solve the heat equation
   essbouncond                # Define where essential boundary conditions are
                              # given (not the value)
                              # See Users Manual Section 3.2.2
      points p1               # left-hand side point
end
#
#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4
#
matrix
   method = 9       # compact matrix, stored per row
                    # necessary for overrelaxation
end

# Define the initial temperature
#  See Users Manual Section 3.2.10

create vector
   value = 10                    # initial Temperature (degree C)
end

# Define the essential boundary conditions
#  See Users Manual Section 3.2.5

essential boundary conditions
   points, p1, value=-20        # boundary Temperature (degree C)
end

#  Define the coefficients for heat equation
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients
   elgrp1(nparm=20)
      coef6  = 1                # kappa in Kirchoff Temperature,
                                # must be 1, see Standard Problems 6.2
      coef17 = 1                # rho*c in enthalpy,
                                # must be 1, see Standard Problems 6.2
end

#  Input for time integration
#  See Users Manual Section 3.2.15

time_integration
   method = euler_implicit      # Integration by Euler implicit
   tinit = $t0                  # initial time
   tend  = $t_end               # end time
   tstep = $dt                  # time step
   toutinit = $t0               # initial time for output
   toutend  = $t_end            # end time for output
   toutstep = {10*$dt}          # time step for output (once in 10 time steps)
```

```
   seq_coefficients = 1          # sequence number for coefficients (default)
   seq_solution_method = 1       # sequence number for linear solver (default)
   mass_matrix = constant        # mass matrix is constant for each time
   stiffness_matrix = constant   # stiffness matrix is constant for each time
   right_hand_side = zero        # no source
end

#  Input for enthalpy integration
#  See Manual Standard Problems Section 6.1

enthalpy_integration
  seq_time_integration = 1       # refers to time integration input (default)
  seq_boundary_conditions = 1    # refers to essential boundary conditions
                                 # default
                                 # All other parameters are given in the block
                                 # constants
end

#  Define which linear solver must be used and what accuracy is required
#  Overrelaxation is used
#  See Users Manual Section 3.2.8

solve, sequence_number = 1
   iteration_method = overrelaxation, omega = 1, max_iter = 1000//
   niter1 = 5, niter2 = 10, print_level= 0   # omega must be reset each time
                              # step
                              # niter1 and niter2 are used to estimate a
                              # value for omega in each step
                              # These values do not have to be optimal
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved

structure
  # Fill initial condition for the temperature
   create_vector, vector %Temperature
  # Compute the initial enthalpy
   compute_enthalpy
  # Write both vectors to sepcomp.out
   output, sequence_number=1

  # Time loop
   start_time_loop
     # Raise time and compute new temperature and enthalpy
      enthalpy_integration
     # Write both vectors to sepcomp.out
      output, sequence_number=1
   end_time_loop
end
```

The postprocessing file is almost identical to the one given in Section 6.1.1.1 and will not be repeated here. Figures 6.1.1.5 to 6.1.1.8 have the same meaning as Figures 6.1.1.1 to 6.1.1.4, but now for the new parameters.

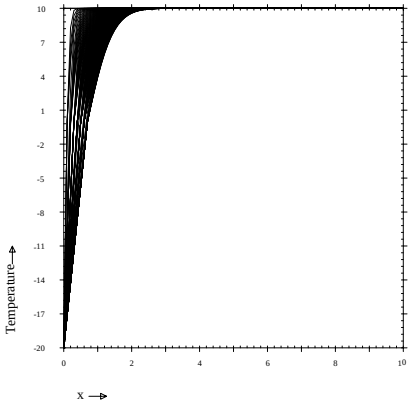


Figure 6.1.1.5: Temperature

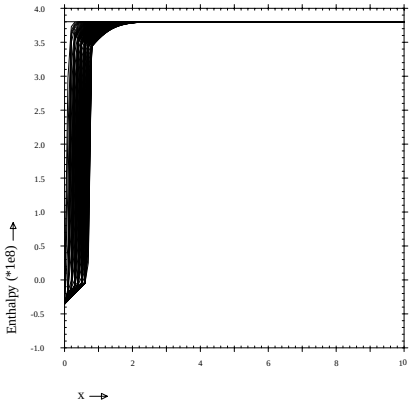


Figure 6.1.1.6: Enthalpy

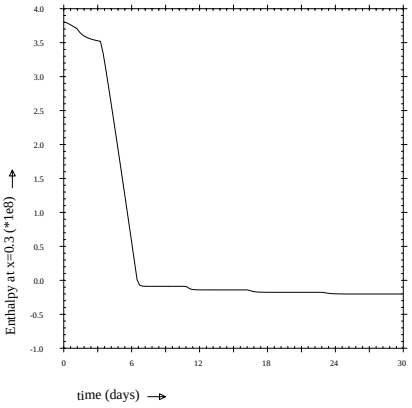


Figure 6.1.1.7: Enthalpy at x=0.3

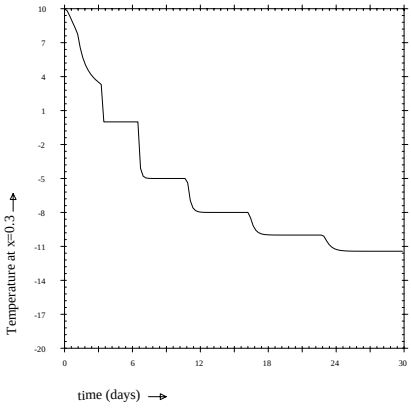


Figure 6.1.1.8: Temperature at x=0.3

6.1.1.3 2D Stefan problem with equal parameters for both phases

This example is completely identical to the one in Section 6.1.1.1. The only difference is that we have a second dimension, but the solution is constant in y-direction.
The mesh is defined by the following mesh input file

```
#  enthalpy2d_1.msh
#
#  mesh file for 2d stefan problem with equal parameters in both phases
#  The enthalpy method is applied
#  See Manual Examples Section 6.1.1.3
#
#  To run this file use:
#     sepmesh enthalpy2d_1.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants
   reals
      length = 10      # length of the region
      width  = 1       # width of the region
   integers
      n = 100          # number of elements in height direction
      m = 25           # number of elements in width direction
      lin = 1          # linear elements
      shape_sur = 3    # triangles
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0, 0)              # Left bottom point
      p2=($length, 0)        # Right bottom point
      p3=($length, $width)   # Right upper point
      p4=(0, $width)         # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Linear elements are used
      c1=line $lin (p1,p2,nelm=$n)     # lower boundary
      c2=line $lin (p2,p3,nelm=$m)     # right boundary
      c3=line $lin (p3,p4,nelm=$n)     # top boundary
      c4=line $lin (p4,p1,nelm=$m)     # left boundary

   surfaces
      s1 = rectangle $shape_sur (c1, c2, c3, c4)

   plot                          # make a plot of the mesh
                                 # See Users Manual Section 2.2
```

```
end
```

The difference in problem file in Section 6.1.1.1 is very small and the reader is referred to the input file in the directory sourceexam.
Finally the postprocessing file has the following contents

```
# enthalpy2d_1.pst
#
#  Input file for postprocessing for 2d stefan problem with equal parameters
#  in both phases
#  The enthalpy method is applied
#  See Manual Examples Section 6.1.1.3
#
#  To run this file use:
#     seppost enthalpy1d_1.pst > enthalpy1d_1.out
#
#  Reads the files meshoutput and sepcomp.out
#
#  Define some general constants
#
constants
   reals
      day = {1/(3600*24)}    # 1/ (number of seconds in a day)
end
#
postprocessing                     # See Users Manual Section 5.2

   time = (0, 2592000,10)   # Do for each ten-th time step
      plot contour V%Temperature
      plot contour V%Enthalpy
   time = 2.592e6

   plot contour V%Temperature
   plot intersection V%Temperature origin = (0, 0.5), angle = 0 //
     texty = 'Temperature at y=0.5', textx = 'x', noplot_legenda

#  Plot the time history of enthalpy and temperature at = (0.3,0.4)
#  The time scale is made in days, so we have to divide the x scale
#  by the number of seconds in a day
#  The enthalpy is scaled by a factor of 10^-8

  time history plot point (0.3,0.4) V%Enthalpy, xscale=$day, factor=1d-8//
     textx = 'time (days)', texty = 'Enthalpy at x=0.3 (*1e8)', noplot_legenda
  time history plot point (0.3,0.4) V%Temperature, xscale=$day//
     textx = 'time (days)', texty ='Temperature at x=0.3', noplot_legenda

end
```

Pictures are comparable to that in Section 6.1.1.1.

6.1.2 Enthalpy approach by quasi-Newton

6.1.2.1 A classical semi-infinite half-space solidification problem

We consider the same example as in Section 6.1.1.1. The only difference is that the problem is solved by the quasi-Newton method of Nedjar, rather than the over-relaxation method.
To get these examples into your local directory use:

```
sepgetex enthalpyxd_y
```

with x and y one-digit numbers.
and to run it use:

```
sepmesh enthalpyxd_y.msh
sepcomp enthalpyxd_y.prb
seppost enthalpyxd_y.pst
```

In case the file `enthalpyxd_y.f` exists this must be replaced by:

```
sepmesh enthalpyxd_y.msh
seplink enthalpyxd_y
enthalpyxd_y < enthalpyxd_y.prb
seppost enthalpyxd_y.pst
```

After the first and last step you may view the results using sepview.

The following values for `x` are available:

```
x = 1, 2
```

and for `y`:

```
y = 3, 4, 5
```

Not all combinations of `x` and `y` have been programmed yet.
x defines the dimension of the space and y the sequence number of the parameter set.

1D Stefan problem with equal parameters for both phases

This is the same example as in Section 6.1.1.1.
The mesh file is almost identical to the one in Section 6.1.1.1.
The reader is referred to the actual input file (`enthalpy1d_3.xxx`) to see the text.

Also the problem file looks very much the same as in Section 6.1.1.1. We show only different parts.

```
# enthalpy1d_3.prb
#
#  problem file for 1d stefan problem with equal parameters in both phases
#  The enthalpy method is applied
#  Solution by quasi-newton
#  See Manual Examples Section 6.1.2.1
#
#  To run this file use:
#     sepcomp enthalpy1d_3.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
    ......
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=810                # Type number for enthalpy equation
                                # solved by quasi-newton
                                # See Standard problems Section 6.1.2
                                # Is also used to solve the heat equation
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      points p1                 # left-hand side point
end
#
#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4
#
matrix
   method = 5       # compact symmetric matrix
end

# Define the initial temperature
#  See Users Manual Section 3.2.10

create vector
```

```
   value = 2                     # initial Temperature (degree C)
   points, p1, value=-4          # boundary Temperature (degree C)
end

# Define the essential boundary conditions
#  See Users Manual Section 3.2.5

essential boundary conditions
   points, p1, value=-4         # boundary Temperature (degree C)
end

#  Define the coefficients for heat equation
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients
   elgrp1(nparm=25)
     icoef3  = 3                 # Type of numerical integration (2 point Gauss)
     icoef5  = %Temperature      # sequence number of temperature vector
      coef6  = $kappa            # thermal conductivity
      coef17 = $rho              # density
      coef18 = $capacity_s       # heat capacity in solid
      coef19 = $capacity_l       # heat capacity in fluid
      coef20 = $latent_heat      # latent heat
      coef21 = $melt_temp        # melting temperature
     icoef22 = %Enthalpy         # sequence number of enthalpy vector
end

#  Input for time integration
#  See Users Manual Section 3.2.15

time_integration
   method = euler_implicit       # Integration by Euler implicit
   tinit = $t0                   # initial time
   tend  = $t_end                # end time
   tstep = $dt                   # time step
   toutinit = $t0                # initial time for output
   toutend  = $t_end             # end time for output
   toutstep = $dt                # time step for output
   seq_solution_method = 1       # sequence number for linear solver (default)
   mass_matrix = constant        # mass matrix is constant for each time
   stiffness_matrix = constant   # stiffness matrix is constant for each time
   right_hand_side = zero        # no source
   abs_iteration_accuracy = 1d-5 # accuracy for non-linear iteration
   max_iter = 1000               # maximum number of non-linear iterations
   print_level = 2               # defines amount of output
   non_linear_iteration          # necessary to activate the non-linear
                                 # iteration per time step
end

#  Input for enthalpy integration
#  See Manual Standard Problems Section 6.1.1

enthalpy_integration
   seq_time_integration = 1       # refers to time integration input (default)
```

```
   solution_method = nedjar       #
   seq_coefficients = 1           # sequence number for coefficients (default)
   seq_boundary_conditions = 1    # refers to essential boundary conditions
                                  # default
                                  # All other parameters are given in the block
                                  # constants
end

#  Define which linear solver must be used and what accuracy is required
#  Conjugate gradients is used with a default preconditioner
#  See Users Manual Section 3.2.8

solve
   iteration_method = cg
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved

structure
  ......
end
```

Also the postprocessing file is almost the same as in Section 6.1.1.1. The only difference is that plotting of the enthalpy is not yet possible. Also the pictures do not show new results.

6.1.2.2 1D Stefan problem with different parameters for both phases

This is the same example as in Section 6.1.1.2.
The mesh is exactly the same as in Section 6.1.1.1
The problem file is a combination of the ones in Sections 6.1.1.2 and 6.1.2.1. See the actual files `enthalpy1d_4.xxx` for the details.
An essential difference is that because the heat conduction is a function of the temperature, we need a function subroutine `funcc3` to compute κ.
See the file `enthalpy1d_4.f`
The postprocessing file is almost identical to the one given in Section 6.1.1.2 and will not be repeated here.
The pictures are a little bit different. They show some extra oscillations compared to the standard method.

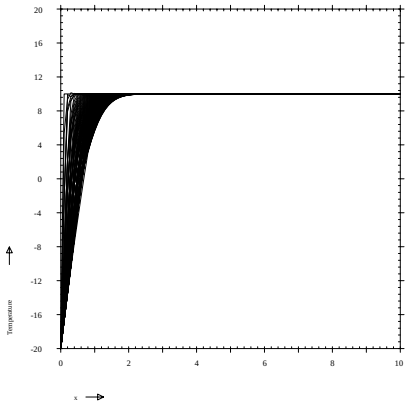


Figure 6.1.2.1: Temperature

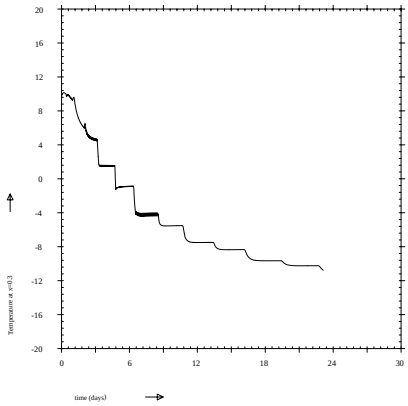


Figure 6.1.2.2: Temperature at x=0.3

6.1.2.3 2D Stefan problem with equal parameters for both phases

This example is completely identical to the one in Section 6.1.1.3. The only difference is that we use the quasi newton method of Nedjar.
The mesh with name `enthalpy2d_2.msh` is almost identical to the mesh file in Section 6.1.1.3.
The problem file `enthalpy2d_2.prb` is a combination of the one in Section 6.1.1.3 and in Section 6.1.2.1 and also the postprocessing file `enthalpy2d_2.pst` is the same as in Section 6.1.1.3.
Pictures are comparable to that in Section 6.1.1.1.

6.1.2.4 1D Stefan problem combined with a heat equation

In this example we consider the combination of a part where we have a melting front and a part of the region where the standard heat equation must be solved. Usually this is of importance in case of different materials but just to show how this works we use the same type of parameters in both parts.
Mark that with the non-linear over-relaxation method the combination of enthalpy equation with standard heat equation is not possible, so we have to use either this quasi newton method or the Newton method treated in Section 6.2.
This example is completely artificial and is in fact identical to the one in Section 6.1.2.1. The only difference is that we have extended the region with a part of the same length in which the heat equation is solved.
To get the corresponding files use

```
sepgetex enthalpy1d_5
```

Below you can find the mesh and problem file without much comment.
The pictures are not very different from pictures shown before.
Mesh file:

```
# enthalpy1d_5.msh
#
#  mesh file for 1d stefan problem with equal parameters in both phases
#  The enthalpy method is applied and in a part of the region only the
#  heat equation solved
#  Solution by quasi-newton
#  See Manual Examples Section 6.1.2.4
#
#  To run this file use:
#     sepmesh enthalpy1d_5.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      length = 1       # length of the region in meters
   integers
      n = 20           # number of elements in phase change part
      m = 20           # number of elements in heat equation part
      lin = 1          # linear elements
end
#
#  Define the mesh
#
mesh1d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1 = 0              # Left-hand point
      p2 = $length        # Right-hand point of phase change region
      p3 = {2*$length}    # Right-hand point
#
#  curves
```

```
#
   curves           # See Users Manual Section 2.3
                    # Linear elements are used
      c1=line $lin (p1,p2,nelm=$n)     # phase change region
      c2=line $lin (p2,p3,nelm=$m)     # heat equation region

#  Since we use two different types of elements, we also need two element groups

   meshline
      lelm1 = (shape=1,c1)     # element group 1: phase change region
      lelm2 = (shape=1,c2)     # element group 2: heat equation region

   plot, nodes = 1                # make a plot of the mesh and plot all nodes
                                  # See Users Manual Section 2.2
end
```

And problem file:

```
# enthalpy1d_5.prb
#
#  problem file for 1d stefan problem with equal parameters in both phases
#  The enthalpy method is applied and in a part of the region only the
#  heat equation solved
#  Solution by quasi-newton
#  See Manual Examples Section 6.1.2.4
#
#  To run this file use:
#     sepcomp enthalpy1d_5.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      kappa = 2                 # thermal conductivity (kg/m^3)
      rho   = 1                 # density  (kg/m^3)
      t0    = 0                 # initial time
      hour  = 3600              # number of seconds in an hour
      day   = {$hour*24}        # number of seconds in a day
      dt    = {6*$hour}         # time step (6h)
      t_end = {100*$day}        # end time (100 days)
      kappa_s = $kappa          # thermal conductivity (solid)
      kappa_l = $kappa          # thermal conductivity (liquid)
      latent_heat = 1e8         # Latent heat (J/kg)
      capacity_s = 2.5d6        # specific heat (solid) (J/kg degree C)
      capacity_l = 2.5d6        # specific heat (liquid)
      melt_temp = 0             # melting temperature (degree C)
   vector_names
      Temperature               # temperature vector
      Enthalpy                  # enthalpy vector
end
```

```
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=810               # Type number for enthalpy equation
                               # solved by quasi-newton
                               # See Standard problems Section 6.1.2
      elgrp2=800               # Type number for second order elliptic equation
                               # See Standard problems Section 3.1
                               # Is used to solve the heat equation
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      points p1,p3             # Both end points
end
#
#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4
#
matrix
   method = 5       # compact symmetric matrix
end

# Define the initial temperature
#  See Users Manual Section 3.2.10

create vector
   value = 1                    # initial Temperature (degree C)
   points, p1, value=-4         # boundary Temperature (degree C)
   points, p3, value=-10        # boundary Temperature (degree C)
end

# Define the essential boundary conditions
#  See Users Manual Section 3.2.5

essential boundary conditions
   points, p1, value=-4         # boundary Temperature (degree C)
   points, p3, value=-10        # boundary Temperature (degree C)
end

#  Define the coefficients for enthalpy and heat equation
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients
   elgrp1(nparm=25)             # enthalpy equation
     icoef3  = 3                # Type of numerical integration (2 point Gauss)
     icoef5  = %Temperature     # sequence number of temperature vector
      coef6  = $kappa           # thermal conductivity
      coef17 = $rho             # density
      coef18 = $capacity_s      # heat capacity in solid
      coef19 = $capacity_l      # heat capacity in fluid
```

```
      coef20 = $latent_heat      # latent heat
      coef21 = $melt_temp        # melting temperature
     icoef22 = %Enthalpy         # sequence number of enthalpy vector
   elgrp2(nparm=25)              # heat equation
     icoef3  = 3                 # Type of numerical integration (2 point Gauss)
      coef6  = $kappa            # thermal conductivity
      coef17 = {$rho*$capacity_s}# rho c_p
end

#  Input for time integration
#  See Users Manual Section 3.2.15

time_integration
   method = euler_implicit       # Integration by Euler implicit
   tinit = $t0                   # initial time
   tend  = $t_end                # end time
   tstep = $dt                   # time step
   toutinit = $t0                # initial time for output
   toutend  = $t_end             # end time for output
   toutstep = $dt                # time step for output
   seq_solution_method = 1       # sequence number for linear solver (default)
   mass_matrix = constant        # mass matrix is constant for each time
   stiffness_matrix = constant   # stiffness matrix is constant for each time
   right_hand_side = zero        # no source
   abs_iteration_accuracy = 1d-5 # accuracy for non-linear iteration
   max_iter = 1000               # maximum number of non-linear iterations
   print_level = 2               # defines amount of output
   non_linear_iteration          # necessary to activate the non-linear
end

#  Input for enthalpy integration
#  See Manual Standard Problems Section 6.1

enthalpy_integration
   seq_time_integration = 1       # refers to time integration input (default)
   solution_method = nedjar       # Defines the Quasi-Newton approach of
                                  # Nedjar
   seq_coefficients = 1           # sequence number for coefficients (default)
   seq_boundary_conditions = 1    # refers to essential boundary conditions
                                  # default
                                  # All other parameters are given in the block
                                  # constants
end

#  Define which linear solver must be used and what accuracy is required
#  Overrelaxation is used
#  See Users Manual Section 3.2.8

solve
   iteration_method = cg
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
```

```
structure
  # Fill initial condition for the temperature
   create_vector, vector %Temperature
  # Compute the initial enthalpy
   compute_enthalpy
  # Write both vectors to sepcomp.out
   output, sequence_number=1

  # Time loop
   start_time_loop
     # Raise time and compute new temperature and enthalpy
      enthalpy_integration
     # Write both vectors to sepcomp.out
      output, sequence_number=1
   end_time_loop
end
```

6.1.2.5 2D Stefan problem combined with a heat equation

This example is the natural extension of the example treated in Section 6.1.2.4.
It concerns a rectangular region of PC material in a large rectangular region consisting of a dielectric. Boundary conditions are only given at the outer boundary of the dielectric. This example shows the behavior of the method for sharp corners. Without showing the pictures we can say that this example shows that in this case a local refinement would lead to much smoother results.
In order to get this example into your directory use:

```
sepgetex enthalpy2d_3
```

The files will not be printed here.

6.1.2.6 2D and 3D Stefan problems with source

These examples concern a region in which initially all material is in a liquid phase. The material has initial temperature 0 (the melting temperature), except for a circle(2D) or sphere(3D), where it varies linearly from 1 in the center to zero at the boundary of the circle. The boundary is kept at a temperature of -2. Due to the boundary condition solidification takes place.
Two examples are available: `enthalpy2d_4` (a square with a circular source) and `enthalpy3d_1` (a 3D block with a spherical source).
To define the linear varying temperature within the source a user function is defined, hence a main program is required.
In order to get these examples into your directory use:

```
sepgetex enthalpyxd_y
```

with xd_y equal to 2d_4 or 3d_1.
To run the examples use:

```
sepmesh enthalpyxd_y.msh
view plots
seplink enthalpyxd_y
enthalpyxd_y < enthalpyxd_y.prb
seppost enthalpyxd_y.pst
```

The files will not be printed here.

6.2 The Newton approach of Fachinotti et al.

This Section is under preparation.

6.3 The heat capacity method

This Section is under preparation.

7 Flow problems

7.1 The isothermal laminar flow of incompressible or slightly compressible liquids

7.1.1 Stationary flow over a backward facing step

As an example of the use of the incompressible flow elements we consider the flow over a backward facing step.
This flow is generally accepted as a benchmark problem used for the comparison of incompressible codes. See Morgan et al for a complete description and results generated by a number of programs. Consider the flow in the backward facing step as demonstrated in Figure 7.1.1.1.

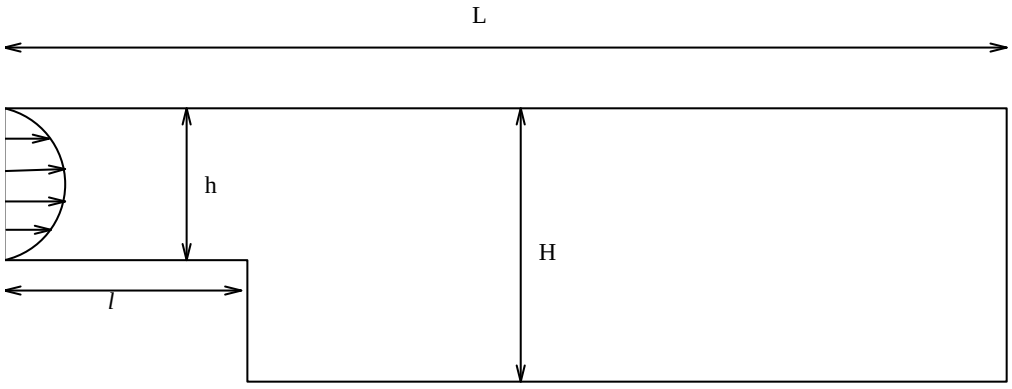


Figure 7.1.1.1: Definition of region for backward facing step

The boundary is subdivided in curves as indicated in Figure 7.1.1.2.

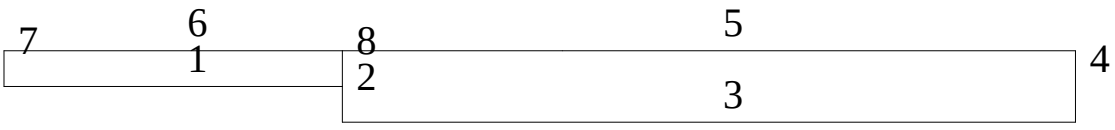


Figure 7.1.1.2: Definition of curves for backward facing step

At the inflow boundary (C7) we assume a quadratic velocity profile with maximum velocity $v_{max} = 1$. The lower wall (C1, C2, C3) and the upper wall (C5 and C6) are fixed, hence a no-slip condition must be prescribed. At the outflow boundary (C4) an outflow boundary condition must be given. This may be for example parallel flow ($\mathbf{u}_t = 0, \sigma_n = 0$) or completely free flow ($\sigma_t = 0, \sigma_n = 0$). Although mathematically incorrect this last boundary condition is the less restrictive and should be used if the end of the outflow region is too close to the step.
Depending on the Reynolds number a recirculation zone arises at the bottom of the step. The Reynolds number is defined as $\mathrm{Re} = u_{max}\frac{H-h}{\eta}$, with

H the width of the outflow pipe.

h the width of the inflow pipe.

l the length of the inflow pipe.

L the sum of the length of inflow and outflow pipe.

Since the flow in inlet and near the outlet is more or less a horizontal flow with a quadratic velocity profile, whereas the flow in the neighborhood of the step shows a recirculation zone, the mesh is refined in the vicinity of the step. In this example the following data are used:

$H = 1$

$h = 0.5$

$l = 6$

$L = 19$

$Re = 50$

To solve this problem we may use a number of solution techniques:

- Penalty method in combination with Crouzeix-Raviart type elements
- Direct (coupled) approach in combination with Crouzeix-Raviart type elements
- Direct (coupled) approach in combination with Taylor-Hood elements.

We shall consider each of these approaches separately.

7.1.1.1 Penalty function approach

The penalty function approach is by far the fastest approach as long as the problem is two-dimensional and the number of elements is not too large. This method is restricted to Crouzeix-Raviart elements only.
In Section 7.1.7 a number of possible elements that can be used is given, but hear we restrict ourselves to quadratic triangles.
In order to get this example into your local directory use

```
sepgetex backwrd2
```

To run the example use

```
sepmesh backwrd2.msh
sepview sepplot.001
sepcomp backwrd2.prb
seppost backwrd2.pst > backwrd2.out
sepview sepplot.001
```

sepmesh requires input from the standard input file:

```
#  backwrd2.msh
#
#  mesh file for backward facing step
#  See Manual Examples Section 7.1.1
#
#  To run this file use:
#     sepmesh backwrd2.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      n_in   =  5       # Number of elements in inlet (flow direction)
      m_step =  5       # Number of elements in step
      m_in   =  5       # Number of elements in inlet (perpendicular to flow)
      n_out  = 20       # Number of elements in outlet (flow direction)
      m_tot  = m_in+m_step # m_in+m_step
      shape_curve = 2   # quadratic elements along the lines
      shape_surf  = 4   # quadratic triangular elements in the surfaces
   reals
      h_wide = 1        # H
      h_step = 0.5      # H-h
      l_in = 6          # l
      l_out = 19        # L
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1 = (0, h_step)            # Lower point of inlet
      p2 = ( l_in, h_step)        # upper point of step
      p3 = ( l_in,0)              # Lower point of step
      p4 = ( l_out,0)             # Lower point of outlet
      p5 = ( l_out, h_wide)       # upper point of outlet
      p6 = ( l_in, h_wide)        # Point above step
      p7 = (0, h_wide)            # upper point of inlet
#
#  curves
#
   curves           # See Users Manual Section 2.3

      # Lower boundary of inlet part
      c1 = line  shape_curve (p1,p2,nelm =  n_in,ratio=1,factor=0.4)
      # step
      c2 = line  shape_curve (p2,p3,nelm =  m_step)
      # Lower boundary of channel
      c3 = line  shape_curve (p3,p4,nelm =  n_out,ratio = 1,factor = 5 )
      # Outlet
      c4 = line  shape_curve (p4,p5,nelm =  m_tot)
```

```
      # right-hand side part of upper boundary
      c5 = translate c3(p6,p5)
      # left-hand side part of upper boundary
      c6 = translate c1(p7,p6)
      # inlet
      c7 = line  shape_curve (p7,p1,nelm =  m_in)
      # artificial line to define 2 surfaces
      c8 = translate c7(p6,p2)
      # left-hand side of channel
      c9 = curves(c8,c2)

#        The next curves are not important for the mesh generation,
#        however, they are used to prescribe the boundary conditions in
#        an easier way

      c20 = curves(c1,c2,c3)                      # lower wall
      c21 = curves(c4)                            # outflow boundary
      c22 = curves(c6,c5)                         # upper wall
      c23 = curves(c7)                            # inlet
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4

     s1 = rectangle  shape_surf (c1,-c8,-c6,c7)     # inlet part
     s2 = rectangle  shape_surf (c3,c4,-c5,c9)      # channel

   plot                     # Plot the mesh

end
```

The parameter `refine_factor` defines how many times the mesh must be refined. If this factor is 1 the standard mesh is used. Is the factor equal to 2, then the number of elements along each of the elements is multiplied by 2, resulting in 4 times the original number of elements.
In order to compute the velocity and pressure program SEPCOMP may be used.
The iteration process is carried out by starting with the Stokes solution, followed by one Picard iteration and followed by Newton iterations.
In this way we get the following input file:

```
# backwrd2.prb
#
#  problem file for backward facing step
#  penalty function approach
#  problem is stationary and non-linear
#  See Manual Examples Section 7.1.1
#
#  To run this file use:
#     sepcomp backwrd2.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
set warn off   ! suppress warnings
#
```

```
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1d-6              # penalty parameter for Navier-Stokes
      rho       = 1                 # density
      eta       = 0.01              # viscosity
   integers
      lower_wall = 20        # curve number for lower wall
      outflow    = 21        # curve number for outflow boundary
      upper_wall = 22        # curve number for upper wall
      inflow     = 23        # curve number for inflow boundary
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=900                # Type number for Navier-Stokes, without swirl
                                # See Standard problems Section 7.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      curves(c lower_wall)      # Fixed under wall (velocity given)
      curves(c upper_wall)      # Fixed upper wall (velocity given)
      degfd2,curves(c outflow) # Outflow boundary (v-component 0)
      curves(c inflow)          # Inflow boundary (velocity given)

end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   # Non-symmetrical profile matrix, So a direct method will be applied
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c inflow), degfd1, quadratic   # The u-component of the velocity at
                                    # instream is quadratic
                                    # The rest of the vector is 0

end

#  Define the coefficients for the problems (first iteration)
```

```
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef6  =  eps        # 6:  Penalty function parameter eps
      coef7  =  rho        # 7:  Density
      coef12 =  eta        #12:  Value of eta (viscosity)
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#

structure                  # See Users Manual Section 3.2.3

  # Compute start vector for the flow by filling boundary conditions
   prescribe_boundary_conditions, velocity

  # Compute the velocity, i.e. solve non-linear problem
   solve_nonlinear_system, velocity

  # Compute the pressure
   derivatives, pressure

  # Write the results to a file
   output
```

```
end

#  The pressure is computed as a derived quantity of the Navier-Stokes
#  equation
#  See Users Manual Section 3.2.11 and Standard Problems Section 7.1

derivatives, sequence_number = 1
   icheld = 7        # means compute pressure
end

#  write the velocity and the pressure to file
#  See Users Manual Section 3.2.13

output
end

end_of_sepran_input
```

To run the program the following steps are performed:

```
    sepcomp backwrd2.prb > backwrd.out
```

If the mesh is refined too much, the buffer length of sepcomp must be enlarged. The procedure to do so is described in the Introduction Manual Section 3.2.
Finally some post-processing actions are carried out by program SEPPOST using the following input file.

```
#  backwrd2.pst
#  Input file for postprocessing for backward facing step
#  See Manual Examples Section 7.1.1
#
#
#  To run this file use:
#     seppost backwrd2.pst > backwrd2.out
#
#  Reads the files meshoutput sepcomp.out
#
#
postprocessing                    # See Users Manual Section 5.2

#  Plot the mesh

   plot mesh

#  Plot the results
#  See Users Manual Section 5.4

   plot identification = text = ' 2D backward facing step ',origin =(10,12)
   plot vector velocity text='velocity field   Re=50'
   plot contour pressure, nlevel = 20 text='pressure contour Re=50'
   3d plot pressure, nlevel=20
   plot coloured levels pressure, nlevel=8

#
#  compute the stream function
```

```
#  See Users Manual Section 5.2
#  store in stream_function

   compute stream function velocity

#  Plot the stream function
#  See Users Manual Section 5.4

   plot contour stream_function, negpos_levels, text='streamlines      Re=50'
   plot contour stream_function, region=(5.5, 10, 0, 1), negpos_levels//
       text='streamlines in the recirculation zone      Re=50'
   plot coloured levels stream_function, negpos_levels

#  Some examples of the use of particle tracking

#  first standard print and plot
   plot track, velocity, pstart = (0,0.6, 0,0.7, 0,0.8, 0,0.9)//
   nmark = 20, tmax = 200, print track
#  next standard plot, print with interpolation and given step
   plot track, velocity, pstart = (0,0.6, 0,0.7, 0,0.8, 0,0.9), nmark = 20//
   tmax = 200, tstep_print = 1, values = (velocity, pressure)
#  finally standard plot, print with interpolation without given step
   plot track, velocity, pstart = (0,0.6, 0,0.7, 0,0.8, 0,0.9)//
   nmark = 20, tmax = 200, values = (velocity)

#  Print of the computed vectors

   print vector velocity
   print vector pressure

end
```

Figure 7.1.1.3 shows the velocity computed and Figure 7.1.1.4 the stream lines. The pressure is shown in Figure 7.1.1.5. Finally Figure 7.1.1.6 shows the streamlines in the recirculation zone. The mesh is too coarse in the neighborhood of the step to get smooth stream lines.

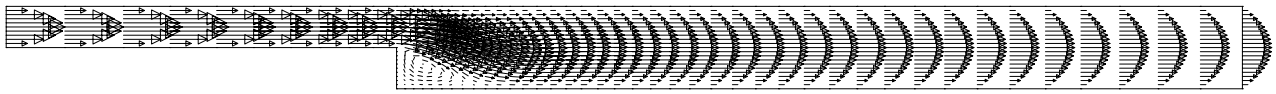


Figure 7.1.1.3: Velocities in backward facing step

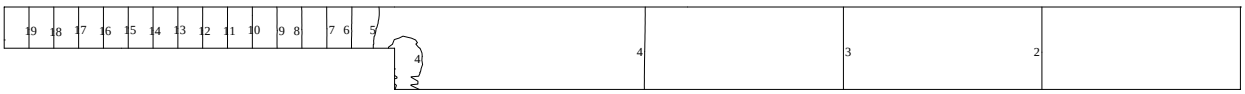


Figure 7.1.1.4: Isobars in backward facing step

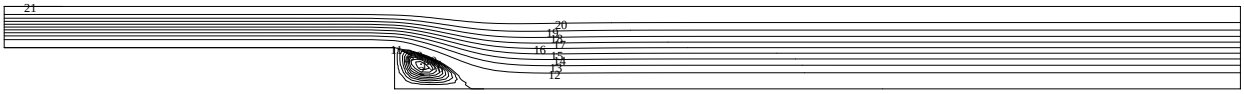


Figure 7.1.1.5: Streamlines in backward facing step

7.1.1.2 Coupled approach

The coupled approach does not need a penalty function parameter and is therefore in general more reliable than the penalty function approach. Unfortunately the coupled approach requires also extra unknowns, since pressure and velocity are solved in one large system of equations. Besides that it is necessary to renumber the unknowns in order to avoid zero diagonal elements. In this example we have combined the coupled approach with an iterative solver for the linear systems of equations. In order to get this example into your local directory use

```
sepgetex backwrd2_it
```

To run the example use

```
sepmesh backwrd2_it.msh
sepview sepplot.001
sepcomp backwrd2_it.prb
seppost backwrd2_it.pst > backwrd2_it.out
sepview sepplot.001
```

The version without iterative linear solver is also available under the name `backwrd2_cp`. To get it locally use

```
sepgetex backwrd2_cp
```

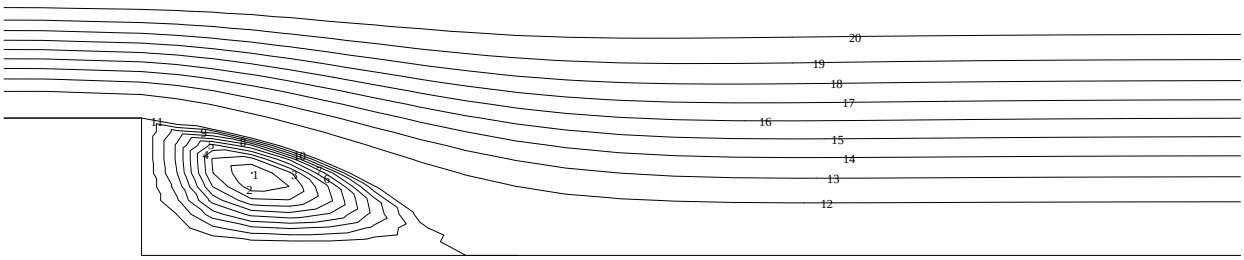


Figure 7.1.1.6: Streamlines in recirculation zone

The mesh file in this case is identical to that of the penalty function approach, except that quadratic triangles with 7 points instead of 6 are used.
The renumbering of the unknowns to avoid zero diagonal elements is done per level since that gives a smaller local band width and in case of an iterative linear solver, usually also a better convergence.
To be sure that there is no possibility of zero diagonal elements in the matrix we have used a very small penalty parameter $\epsilon = 10^{-10}$, which does not influence the solution at all, but puts elements of order 10^{-10} on the main diagonal for the rows corresponding to the continuity equation.
The solution of the linear systems with iterative solvers poses extra complications. Due to the stretching of the elements, we have a large aspect ratio (i.e. ratio of the length and width of elements). The effect is that the linear solver has great difficulties to converge of does not converge at all. In order to be able to apply the iterative solver, it was necessary to combine an ILU preconditioner with extra fill in. This produces a larger matrix with many more "non-zero" elements, where we mean by "non-zero" an element that is stored in the matrix. It may become non-zero when the ILU preconditioning is applied. The only alternative is to decrease the aspect ratio. If the aspect ratio is larger, even in this case no convergence could be reached.
Another problem is that the Newton linearization may produce smaller diagonal elements, which also may influence the convergence dramatically. To that end we use a Picard type linearization in each step of the non-linear iteration process.
Combining all these aspects results in the following input file for the program sepcomp.

```
# backwrd2_it.prb
#
#  problem file for backward facing step
#  direct (coupled) approach
#  problem is stationary and non-linear
#  An iterative linear solver is applied
#  See Manual Examples Section 7.1.1
#
#  To run this file use:
#     sepcomp backwrd2_it.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
set warn off   ! suppress warnings
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
```

```
   reals
      eps       = 1d-10             # penalty parameter for Navier-Stokes
                                    # This parameter is used only to avoid
                                    # zero diagonals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
   integers
      lower_wall = 20        # curve number for lower wall
      outflow    = 21        # curve number for outflow boundary
      upper_wall = 22        # curve number for upper wall
      inflow     = 23        # curve number for inflow boundary
   vector_names
      velocity_pressure      # velocity and pressure are stored in
                             # one solution vector
                             # The pressure is only available in the
                             # centroid
      pressure               # Pressure in the vertices
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=902                # Type number for Navier-Stokes, without swirl
                                # Coupled approach
                                # See Standard problems Section 7.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      curves(c lower_wall)      # Fixed under wall (velocity given)
      curves(c upper_wall)      # Fixed upper wall (velocity given)
      degfd2,curves(c outflow) # Outflow boundary (v-component 0)
      curves(c inflow)          # Inflow boundary (velocity given)

   renumber levels (1,2),(3,4,5)  # The unknowns are renumbered per level in
                                  # order to ensure that first some velocities
                                  # are eliminated before pressures are started
                                  # In this way zero elements at the main
                                  # diagonal are removed by elimination

end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4
matrix
   storage_scheme = compact, extra_fillin = 2  # Non-symmetrical compact matrix
                                  # So an iterative linear solver will be applied
                                  # For convergence of the iterative method we
                                  # need extra fill in
end

# Create start vector and put the essential boundary conditions into this
# vector
```

```
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c inflow), degfd1, quadratic   # The u-component of the velocity at
                                         # instream is quadratic
                                         # The rest of the vector is 0

end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef6  =  eps        # 6:  Penalty function parameter eps
      coef7  =  rho        # 7:  Density
      coef12 =  eta        #12:  Value of eta (viscosity)
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
                             # In case of an iterative linear solver it is
                             # necessary to use Picard instead of Newton
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations
   global_options, maxiter=10, accuracy=1d-2,print_level=2, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#
```

```
structure                  # See Users Manual Section 3.2.3

  # Compute start vector for the flow by filling boundary conditions
   prescribe_boundary_conditions, sequence_number=1, velocity_pressure

  # Compute the velocity, i.e. solve non-linear problem
   solve_nonlinear_system, velocity_pressure

  # Compute the pressure
   derivatives, pressure

  # Write the results to a file
   output
end

#  The pressure is computed as a derived quantity of the Navier-Stokes
#  equation
#  See Users Manual Section 3.2.11 and Standard Problems Section 7.1

derivatives
   icheld = 7        # means compute pressure
   seq_input_vector = velocity_pressure
end

solve
   iteration_method = bicgstab, accuracy = 1d-2, print_level = 2 //
   start = old_solution, preconditioning = ilu
end

end_of_sepran_input
```

Results of the computation are almost the same as for the penalty function method and are not repeated here. Of course the post processing file is the same as for the penalty function method.

7.1.1.3 Coupled approach with Taylor-Hood elements

The usage of Taylor-Hood elements is almost the same as for the coupled approach. The only difference is that now the pressure is defined in vertices of the elements. This gives a slight difference in the problem file. Available are the quadratic Taylor-Hood triangles (backwrd2_th) and the linear Taylor-Hood triangles, the so-called mini element (backwrd2_mini). To get these examples into your local directory use

```
sepgetex backwrd2_xx
```

with `xx` either `th` or `mini`. As illustration we give here the quadratic problem file.

```
# backwrd2_th.prb
#
#  problem file for backward facing step
#  direct (coupled) approach using Taylor-Hood elements
#  problem is stationary and non-linear
#  A direct linear solver is applied
#  See Manual Examples Section 7.1.1
#
#  To run this file use:
#     sepcomp backwrd2_th.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
set warn off   ! suppress warnings
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1d-10             # penalty parameter for Navier-Stokes
                                    # This parameter is used only to avoid
                                    # zero diagonals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
   integers
      lower_wall = 20        # curve number for lower wall
      outflow    = 21        # curve number for outflow boundary
      upper_wall = 22        # curve number for upper wall
      inflow     = 23        # curve number for inflow boundary
   vector_names
      velocity_pressure      # velocity and pressure are stored in
                             # one solution vector
end
#
#  Define the type of problem to be solved
#
problem                  # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
```

```
      elgrp1=903               # Type number for Navier-Stokes, without swirl
                               # Coupled approach
                               # See Standard problems Section 7.1
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      degfd1, degfd2, curves(c lower_wall)     # Fixed under wall
                               # (velocity given, not the pressure)
      degfd1, degfd2, curves(c upper_wall)     # Fixed upper wall
                               # (velocity given, not the pressure)
      degfd2,curves(c outflow) # Outflow boundary (v-component 0)
      degfd1, degfd2, curves(c inflow)   # Inflow boundary (velocity given)

   renumber levels (1,2),(3)      # The unknowns are renumbered per level in
                                  # order to ensure that first some velocities
                                  # are eliminated before pressures are started
                                  # In this way zero elements at the main
                                  # diagonal are removed by elimination

end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   # Non-symmetrical profile matrix, So a direct method will be applied
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c inflow), degfd1, quadratic   # The u-component of the velocity at
                                  # instream is quadratic
                                  # The rest of the vector is 0

end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef6  =  eps        # 6:  Penalty function parameter eps
      coef7  =  rho        # 7:  Density
      coef12 =  eta        #12:  Value of eta (viscosity)
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7
```

```
change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations
   global_options, maxiter=10, accuracy=1d-2,print_level=2, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#

structure                  # See Users Manual Section 3.2.3

  # Compute start vector for the flow by filling boundary conditions
   prescribe_boundary_conditions, velocity_pressure

  # Compute the velocity, i.e. solve non-linear problem
   solve_nonlinear_system, velocity_pressure

  # Write the results to a file
   output
end

end_of_sepran_input
```

7.1.1.4 Time dependent approach with Taylor-Hood elements

Another way to solve the stationary equations is by solving it as the limit of a time-dependent problem. So we start with a zero velocity (except for the boundary conditions) and solve the instationary equations. As time proceeds the solution approaches the stationary solution.
This example is called `backwrd2_thinst`.
In order to get this example into your local directory use.

```
sepgetex backwrd2_thinst
```

To run the example use

```
sepmesh backwrd2_thinst.msh
sepview sepplot.001
sepcomp backwrd2_thinst.prb
seppost backwrd2_thinst.pst > backwrd2_thinst.out
sepview sepplot.001
```

Only the problem file differs essentially from the ones previously treated. This file is given by

```
# backwrd2_thinst.prb
#
#  problem file for backward facing step
#  direct (coupled) approach using Taylor-Hood elements
#  problem is stationary and non-linear, but is solved instationary
#
#  An iterative linear solver is applied
#  See Manual Examples Section 7.1.1
#
#  To run this file use:
#     sepcomp backwrd2_thinst.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
set warn off   ! suppress warnings
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1d-10             # penalty parameter for Navier-Stokes
                                    # This parameter is used only to avoid
                                    # zero diagonals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
      t0        = 0                 # initial time
      dt        = 0.1               # time step
      tend      = 5                 # end time
      tout0     =  t0               # First time that a result is written
      toutend   =  tend             # End time for writing
      toutstep  = 5*dt            # In each 5^th time step the result is written
   integers
```

```
      outflow    = 21        # curve number for outflow boundary
      wall       = 25        # curve number for walls
      inflow     = 23        # curve number for inflow boundary
   vector_names
      velocity_pressure      # velocity and pressure are stored in
                             # one solution vector
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=903               # Type number for Navier-Stokes, without swirl
                               # Coupled approach
                               # See Standard problems Section 7.1
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      degfd1, degfd2, curves(c wall)     # Fixed wall
                               # (velocity given, not the pressure)
      degfd2,curves(c outflow) # Outflow boundary (v-component 0)
      degfd1, degfd2, curves(c inflow)   # Inflow boundary (velocity given)

   renumber levels (1,2),(3)      # The unknowns are renumbered per level in
                                  # order to ensure that first some velocities
                                  # are eliminated before pressures are started
                                  # In this way zero elements at the main
                                  # diagonal are removed by elimination

end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   storage_scheme = compact  # Non-symmetrical compact matrix
                             # So an iterative linear solver will be applied
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c inflow), degfd1, quadratic   # The u-component of the velocity at
                                    # instream is quadratic
                                    # The rest of the vector is 0

end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
```

```
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 1           # 5:  Type of linearization (1=Picard)
      coef6  =  eps        # 6:  Penalty function parameter eps
      coef7  =  rho        # 7:  Density
      coef12 =  eta        #12:  Value of eta (viscosity)
end

#  Definition of time integration
#  See Users Manual Section 3.2.15

time_integration
   method = euler_implicit         # Integration by the Euler implicit method
   tinit =  t0                     # Initial time
   tend =  tend                    # End time
   tstep =  dt                     # Time step
   toutinit =  tout0               # First time that a result is written
   toutend =  toutend              # End time for writing
   toutstep =  toutstep            # time steps for writing
   boundary_conditions = constant  # The boundary conditions do not depend on
                                   # time
   seq_boundary_conditions = 1     # Sequence number for the input of the
                                   # essential boundary conditions
   seq_coefficients = 1            # Sequence number for the coefficients
   seq_output = 1                  # Sequence number for the output
   mass_matrix = constant          # Time-independent mass matrix
   number_of_coupled_equations = 1 # There is only one equation

end

#  input for the linear solver
#  See Users Manual Section 3.2.8

solve
   iteration_method = cg, preconditioner = ilu, print_level = 1
end

#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#

structure                  # See Users Manual Section 3.2.3

  # Compute start vector for the flow by filling boundary conditions
   prescribe_boundary_conditions, sequence_number=1, velocity_pressure

  # Time loop
   start_time_loop

   # One time step to compute the velocity
     time_integration, velocity_pressure
```

```
     output, sequence_number=1

   end_time_loop

end

end_of_sepran_input
```

7.1.2 Stationary isothermal non-Newtonian flow in a T-shaped region using the penalty function method

In this example we consider the non-Newtonian flow in a channel in a t-configuration (Cartesian co-ordinates). In fact this is the same example as in the Introduction Section 7.3, however with a non-Newtonian model instead of the Newtonian model. The region of definition has the same shape as in Figure 7.3.3 in the Introduction, however, with slightly different co-ordinates. The boundary conditions are taken exactly the same as in the Introduction.
As viscosity model a power law model with $\eta_n = 0.1$ and $n = 0.5$ is used. The penalty parameter ε is equal to 10^{-6}.

The iteration process starts with the Stokes equation (MCONV=0), the second iteration is performed by Picard iteration (MCONV=1), and the succeeding iterations by the Newton method (MCONV=2).
To increase the convergence of the iteration process for the non-Newtonian iteration process it is useful to take an overrelaxation parameter ω of the shape: $\omega = 1 + \beta\,(1-n)$ with n the power in the Power law model. Tanner et al (1975) have shown that $\beta \approx 0.4$ gives satisfactory results. Therefore in the program $relaxation = 1.2$ is used.

The mesh input file for this example is:

```
* tshapenn.msh
mesh2d

   points

      p1=(0,0)
      p2=(3,0)
      p3=(20,0)
      p4=(20,3)
      p5=(3,3)
      p6=(3,20)
      p7=(0,20)
      p8=(0,3)

   curves

*
* Fixed under wall:     C1, C2
* Outstream boundary:   C3
* Fixed side walls:     C4, C5
* Instream boundary:    C6
* Symmetry axis:        C7, C8
*
*  Straight lines with equidistant grid:  C1, C3, C6, C8, C9, C10
*  Straight lines with graded grid:  C2, C4, C5
*
      c1=line2(p1,p2,nelm=4)
      c2=line2(p2,p3,nelm=8,ratio=1,factor=3)
      c3=line2(p3,p4,nelm=4)
      c4=line2(p4,p5,nelm=8,ratio=3,factor=3)
      c5=line2(p5,p6,nelm=8,ratio=1,factor=3)
      c6=line2(p6,p7,nelm=4)
      c7=line2(p7,p8,nelm=8,ratio=3,factor=3)
      c8=line2(p8,p1,nelm=4)
      c9=line2(p8,p5,nelm=4)
```

```
      c10=line2(p2,p5,nelm=4)

   surfaces

*  The surfaces are generated by QUADRILATERAL in order to get a rectangular
*  grid

      s1=quadrilateral4(c1,c10,-c9,c8)
      s2=quadrilateral4(c2,c3,c4,-c10)
      s3=quadrilateral4(c5,c6,c7,c9)

*  Plot the mesh:

   plot

end
```

In order to compute the velocity and pressure program SEPCOMP may be used. The iteration process is carried out by starting with the Stokes solution, followed by one Picard iteration and followed by Newton iterations.
In this way we get the following input file:

```
* tshapenn.prb
set warn off    ! suppress warnings
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   vector_names
      velocity
      pressure
end

problem
   # Define type of elements
   types
   elgrp1=900               # Type number for Navier-Stokes, without swirl
                            # 6-point triangle
                            # Approximation 7-point extended triangle
                            # Penalty function method
   # Define where essential boundary conditions are present
   essbouncond
      curves(c1,c2)        # Fixed under wall
      curves(c4,c6)        # Fixed side walls and instream boundary
      degfd1=curves(c7,c8) # Symmetry axis (only u-component)

end

* define type of matrix

matrix
   # Non-symmetrical profile matrix, So a direct method will be applied
end

* Create start vector and put the essential boundary conditions into this
```

```
* vector

essential boundary conditions
   value = 0                       # First set vector equal to zero

   # Next fill all non-zero essential boundary conditions
   curves(c6), degfd2, value = -1  # The v-component of the velocity at
                                   # instream is -1

end

*  Define coefficients for the first iteration

coefficients
   elgrp1 ( nparm=20)     # The coefficients are defined by 8 parameters
   icoef2 = 2             # 2:  type of constitutive equation (2=Power-law)
   icoef5 = 0             # 5:  Type of linearization (0=Stokes flow)
   coef6  = 1d-6          # 6:  Penalty function parameter eps
   coef7  = 1             # 7:  Density
                          # 8:  angular velocity = 0
                          # 9:  body force in x-direction = 0
                          #10:  body force in y-direction = 0
   coef12 = 0.1           #12:  Value of etha_n (viscosity)
   coef13 = 0.5           #13:  Viscosity parameter n
end

*  Define the coefficients for the next iterations

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 3:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 3:  Type of linearization (2=Newton iteration)
end

* Define the parameters for the non-linear solver

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1//
                   relaxation=1.2
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

* Define output, and compute pressure

output
   v1 = icheld=7   # pressure
end
```

```
end_of_sepran_input
```

Finally some post-processing actions are carried out by program SEPPOST using the following input file.

```
* tshapenn.pst

post processing

*  Print both vectors completely

   print velocity
   print pressure

*  Compute stream funnction, store in stream_function, and name this vector

   compute stream_function = stream function velocity

* PLot the results

   plot vector velocity                # Vector plot of velocity
   plot contour pressure               # Contour plot of pressure
   plot contour stream_function        # Contour plot of stream function

end
```

Figure 7.1.2 shows the velocity computed and Figure 7.1.2 the stream lines. The pressure is shown in Figure 7.1.2.

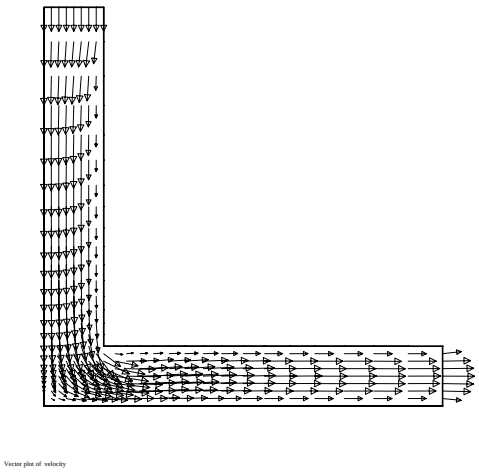


Figure 7.1.2.1: Vector plot of velocity in flow problem

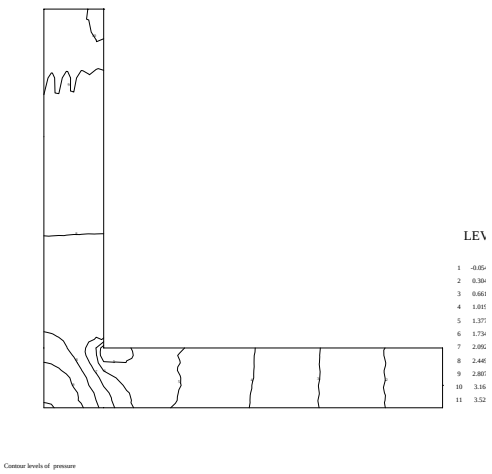


Figure 7.1.2.2: Isobars in flow problem

7.1.3 Stationary isothermal Newtonian flow in a T-shaped region using the integrated solution method

In this example we consider the Newtonian flow in a channel in a t-configuration (cartesian co-ordinates). In fact this is the same example as in the Section 7.1.2, however with a Newtonian model instead of the Non-newtonian model. The region of definition has the same shape as in Figure 7.3.3 in the Introduction, however, with slightly different co-ordinates. The boundary conditions are taken exactly the same as in the Introduction.
The viscosity model is the standard Newtonian model.
Instead of the penalty function method the (direct) integrated solution method is used, which implies that pressure and velocity are computed in a coupled way.
Furthermore the bi-linear quadrilateral elements with shape number 9 are used. In these elements the velocities are defined in the vertices of the elements and the pressure is a constant per element. The corresponding unknown is positioned in the centroid of the element.
This element does not satisfy the so-called Brezzi-Babuška condition (Cuvelier et al, 1986). However, at the outflow we do not describe the normal velocity component and for this specific element this means that the element is still admissible.
The iteration process starts with the Stokes equation (MCONV=0), the second iteration is performed by Picard iteration (MCONV=1), and the succeeding iterations by the Newton method (MCONV=2).

The mesh input file for this example is:

```
* tshapedr.msh
mesh2d

   points

      p1=(0,0)
      p2=(1,0)
      p3=(10,0)
      p4=(10,1)
      p5=(1,1)
      p6=(1,10)
      p7=(0,10)
```

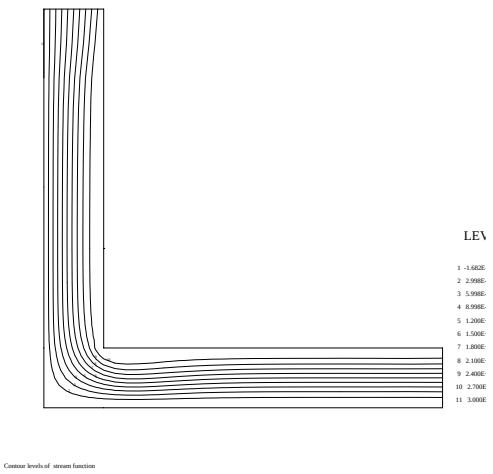


Figure 7.1.2.3: Stream line plot in flow problem

```
      p8=(0,1)

   curves

*    Fixed under wall:     C1, C2
*    Outstream boundary:   C3
*    Fixed side walls:     C4, C5
*    Instream boundary:    C6
*    Symmetry axis:        C7, C8
*
*     Straight lines with equidistant grid:  C1, C3, C6, C8, C9, C10
*     Straight lines with graded grid:  C2, C4, C5
*
      c1=line2(p1,p2,nelm=8)
      c2=line2(p2,p3,nelm=16,ratio=1,factor=3)
      c3=line2(p3,p4,nelm=8)
      c4=translate c2 (p5,p4)
      c5=line2(p5,p6,nelm=16,ratio=1,factor=3)
      c6=translate c1 (p7,p6)
      c7=translate c5 (p8,p7)
      c8=translate c3 (p1,p8)
      c9=translate c1 (p8,p5)
      c10=translate c3 (p2,p5)

   surfaces

*     The surfaces are generated by QUADRILATERAL in order to get a
*     rectangular grid

      s1=quadrilateral9(c1,c10,-c9,-c8)
      s2=quadrilateral9(c2,c3,-c4,-c10)
      s3=quadrilateral9(c5,-c6,-c7,c9)

*     Plot the mesh:
```

```
   plot
   renumber start = c3
end
```

Mark that in this example we have given an explicit start for the renumbering procedure. Experiments have shown that starting at the small side (in this case the outflow) considerably decreases the computation time.

In order to compute the velocity and pressure program SEPCOMP may be used. The iteration process is carried out by starting with the Stokes solution, followed by one Picard iteration and followed by Newton iterations.
Since the integrated solution method is applied, it is necessary to reorder the unknowns such that it is guaranteed that the first unknowns are velocities and not pressures.
In combination with a direct solver this is only efficient if renumbering per level is applied.
In this way we get the following input file:

```
*  tshapedr.prb
*
*
set warn off    ! suppress warnings
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   vector_names
      velocity
      pressure
end

problem
   # Define type of elements
   types
   elgrp1=902               # Type number for Navier-Stokes, without swirl
   # Define where essential boundary conditions are present
   essbouncond
      curves(c1,c2)        # Fixed under wall
      curves(c4)           # Fixed side wall
      curves(c5)           # Fixed side wall
      curves(c6)           # instream boundary
      degfd1=curves(c7)    # Symmetry axis (only u-component)
      degfd1=curves(c8)    # Symmetry axis (only u-component)
      degfd2=curves(c3)    # Outstream boundary (v-component given)
                            # All not prescribed boundary conditions satisy
                            # corresponding stress is zero, i.e.
                            # Tangential stress at C7, C8
                            # Normal stress at C3
   renumber levels (1,2), 3 # For each level, first the velocities and then
                            # the pressure
end

* define type of matrix

matrix
   # Non-symmetrical profile matrix, So a direct method will be applied
end
```

```
* Create start vector and put the essential boundary conditions into this
* vector

essential boundary conditions
   value = 0                       # First set vector equal to zero

   # Next fill all non-zero essential boundary conditions
   curves(c6), degfd2, value = -1  # The v-component of the velocity at
                                   # instream is -1

end

*  Define coefficients for the first iteration

coefficients
   elgrp1 ( nparm=20)     # The coefficients are defined by 8 parameters
   icoef2 = 1             # 2:  type of constitutive equation (1=Newton)
   icoef5 = 0             # 5:  Type of linearization (0=Stokes flow)
   coef7  = 1             # 7:  Density
                          # 8:  angular velocity = 0
                          # 9:  body force in x-direction = 0
                          #10:  body force in y-direction = 0
   coef12 = 0.01          #12:  Value of etha (viscosity)
end

*  Define the coefficients for the next iterations

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 3:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 3:  Type of linearization (2=Newton iteration)
end

* Define the parameters for the non-linear solver

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=2, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

* Define output, and average the pressure

output
   v1 = icheld=7   # averaged pressure
end
end_of_sepran_input
```

Although the pressure is already computed in the integrated method, this pressure is discontinuous over the elements. In order to be able to make contour plots the pressure is averaged and new values in the vertices are computed.

Finally some post-processing actions are carried out by program SEPPOST using the following input file.

```
* tshapedr.pst
postprocessing

*  Print both vectors completely

   print velocity
   print pressure

*  Compute stream function, store in stream_function, and name this vector

   compute stream_function = stream function velocity

* Plot the results

   plot vector velocity                # Vector plot of velocity
   plot contour pressure               # Contour plot of pressure
   plot contour stream_function        # Contour plot of stream function

end
```

Figure 7.1.3.1 shows the velocity computed and Figure 7.1.3.2 the isobars. The stream lines are shown in Figure 7.1.3.3.

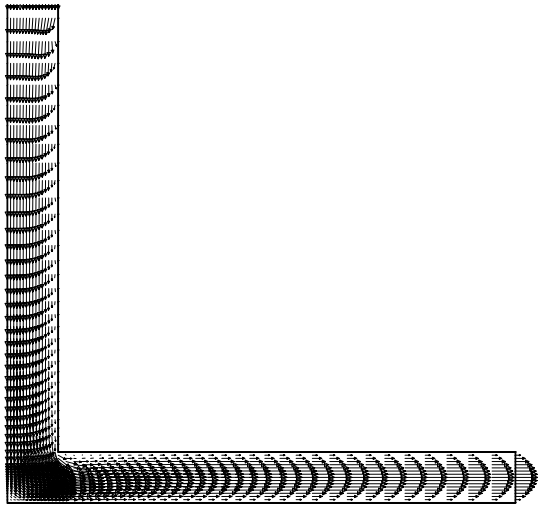


Figure 7.1.3.1: Vector plot of velocity in flow problem

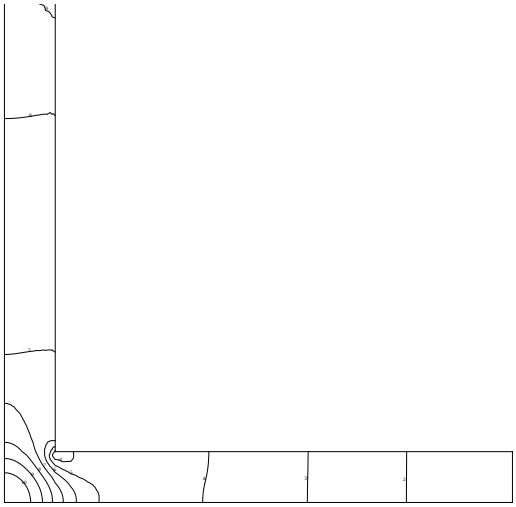


Figure 7.1.3.2: Isobars in flow problem

7.1.4 Stationary flow over a 3D backward facing step using the integrated solution method

In this example we consider a three-dimensional example of a stationary flow. Since three-dimensional problems are usually too large to be solved by a direct linear solver, this example is combined with an iterative method. This automatically implies that we can not use the penalty function method, since the matrix produced by the penalty function method is very ill-conditioned and no iterative solver is able to converge. Hence the integrated approach is applied.

The example we consider is the natural extension of the 2D backward facing step shown in example 7.1.1. Figure 7.1.1.1 shows the cross-section of the region in the y is constant plane. From the results in Section 7.1.1 it is clear that we may take a smaller inlet and outlet to get comparable results in the vicinity of the step. In order to get this example into your local directory use

```
sepgetex backwrd3
```

To run the example use

```
sepmesh backwrd3.msh
sepview sepplot.001
seplink backwrd3
backwrd3 < backwrd3.prb
seppost backwrd3.pst > backwrd3.out
sepview sepplot.001
```

To create the mesh, we first have to define the points, curves, surfaces and volumes. Figure 7.1.4.1 shows the points, curves and surfaces of the front plane. The curves C5 and C6 are clustered to a new curve C10 and the curves C1, C2 and C3 to a new curve C11. The surfaces S1 and S2 are clustered to a surface S3.
The back plane S4 is just a translation of S3, where the curves are translated as follows:
C11: C12, C4: C13, C10: C14 and C7: C15.
The total volume is considered as a pipe. The front and back surfaces are considered as bottom and top surface of this pipe respectively and the other 4 surfaces as parts of a pipe surface. These

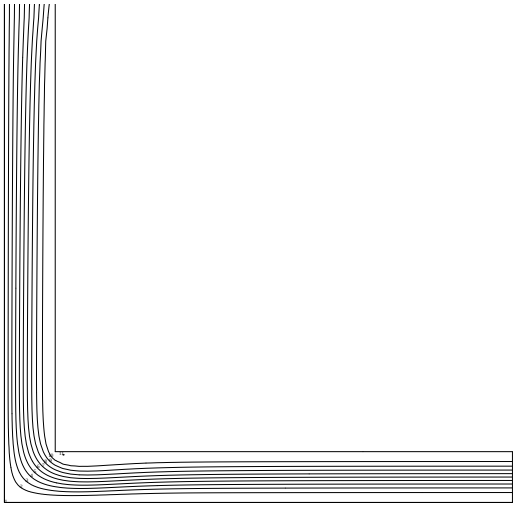


Figure 7.1.3.3: Stream line plot in flow problem

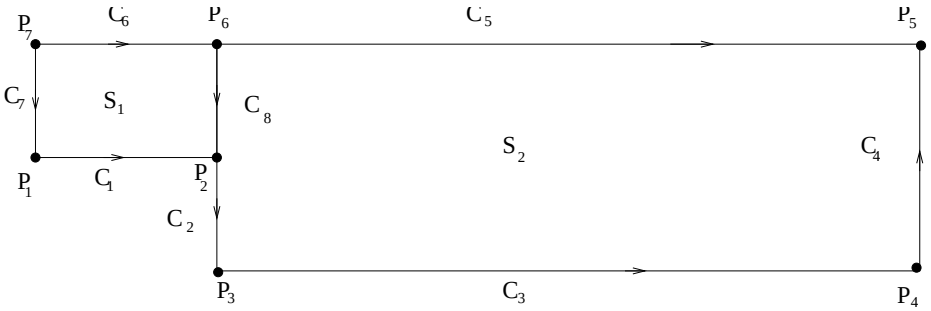


Figure 7.1.4.1: Definition of front surface for 3D backward facing step

4 subsurfaces are sketched in Figure 7.1.4.2. In the y-direction we have a constant thickness of 1. Figure 7.1.4.3 shows a plot of all curves. At the inflow boundary (S5) we assume a quadratic velocity profile with maximum velocity $v_{max} = 1$. The lower wall (S6) and the upper wall (S8) and the side walls (S3 and S4) are fixed, hence a no-slip condition must be prescribed. At the outflow boundary (S7), an outflow boundary condition must be given. For the same reason as in Example 7.1.1 we choose for a completely free flow.
Depending on the Reynolds number a recirculation zone arises at the bottom of the step. The Reynolds number is defined as $\mathrm{Re} = u_{max}\frac{H-h}{\eta}$, with

H the width of the outflow pipe.

h the width of the inflow pipe.

l the length of the inflow pipe.

L the sum of the length of inflow and outflow pipe.

Since the flow in inlet and near the outlet is more or less a horizontal flow with a quadratic velocity profile, whereas the flow in the neighborhood of the step shows a recirculation zone, the mesh is refined in the vicinity of the step. In this example the following data are used:

$H = 1$

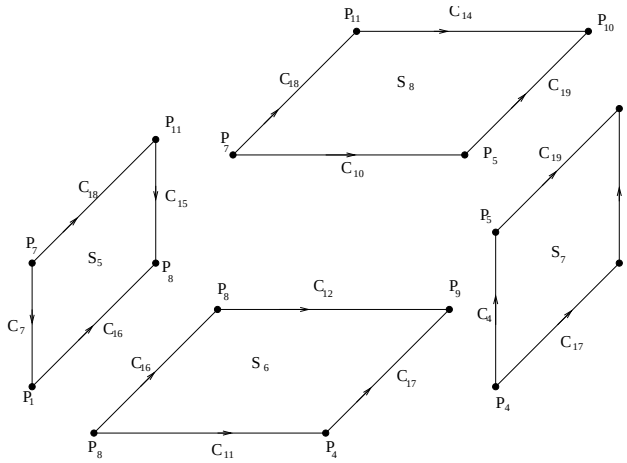


Figure 7.1.4.2: Definition of pipe surfaces for 3D backward facing step

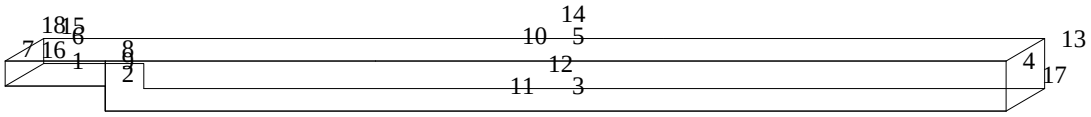


Figure 7.1.4.3: Definition of curves for 3D backward facing step

$h = 0.5$

$l = 2$

$L = 20$

$Re = 50$

The mesh is generated by program sepmesh. The elements used are quadratic hexahedrons with 27 points per element.
sepmesh requires input from the standard input file:

```
* backwrd3.msh
*
*   Mesh for 3D backward facing step as defined in
*   manual Standard Problems Section 7.1.4

constants
   integers
      n_in   =  5       # Number of elements in inlet (flow direction)
      m_step =  5       # Number of elements in step
      m_in   =  5       # Number of elements in inlet (perpendicular to flow)
      n_out  = 20       # Number of elements in outlet (flow direction)
      m_tot  = m_in+m_step  # m_in+m_step
      n_y    =  5       # Number of elements in y-direction
   reals
```

```
      h_wide = 1        # H
      h_step = 0.5      # H-h
      l_in =  2        # l
      l_out =20         # L
      y_min = 0         # ymin
      y_max = 1         # ymax
end
mesh3d
   points
      p1 = (0, y_min, h_step)
      p2 = ( l_in, y_min, h_step)
      p3 = ( l_in, y_min,0)
      p4 = ( l_out, y_min,0)
      p5 = ( l_out, y_min, h_wide)
      p6 = ( l_in, y_min, h_wide)
      p7 = (0, y_min, h_wide)
      p8 = (0, y_max, h_step)
      p11= (0, y_max, h_wide)
   curves
      c1 = line2(p1,p2,nelm =  n_in,ratio=1,factor=0.4)
      c2 = line2(p2,p3,nelm =  m_step)
      c3 = line2(p3,p4,nelm =  n_out,ratio = 1,factor = 5 )
      c4 = line2(p4,p5,nelm =  m_tot)
      c5 = translate c3(p6,p5)
      c6 = translate c1(p7,p6)
      c7 = line2(p7,p1,nelm =  m_in)
      c8 = translate c7(p6,p2)
      c9 = curves(c8,c2)
      c10= curves(c6,c5)
      c11= curves(c1,c2,c3)
      c12= translate c11 (p8,-p9)
      c13= translate c4  (p9,p10)
      c14= translate c10 (p11,-p10)
      c15= translate c7  (p11,p8)
      c16= line2 (p1,p8,nelm= n_y)
      c17= translate c16 (p4,p9)
      c18= translate c16 (p7,p11)
      c19= translate c16 (p5,p10)
   surfaces
     s1 = rectangle6(c1,-c8,-c6,c7)
     s2 = rectangle6(c3,c4,-c5,c9)
     s3 = surfaces(s1,s2)
     s4 = translate s3 ( c12, c13,-c14, c15 )
     s5 = pipesurface 6 ( c7 , c15, c18, c16 )
     s6 = pipesurface 6 ( c11, c12, c16, c17 )
     s7 = pipesurface 6 ( c4 , c13, c17, c19 )
     s8 = pipesurface 6 (-c10,-c14, c19, c18 )
     s9 = ordered surface ( s6,s7,s8,s5)
   volumes
     v1 = pipe14 ( s3, s4, s9 )
   plot, eyepoint = (50,-10,5)
end
```

To create the mesh the following steps are performed:

```
 sepmesh < backwrd3.msh
```

```
sepview
```

Figure 7.1.4.4 shows the final mesh. In order to compute the velocity and pressure program SEP-

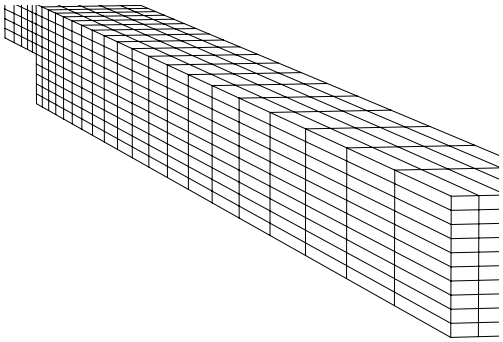


Figure 7.1.4.4: Mesh for 3D backward facing step

COMP may be used. Since the inflow velocity depends on the space, a function subroutine is necessary for the essential boundary conditions. Furthermore for this 3D problem we need a larger buffer. For that reason sepcomp is replaced by program backwrd3.f.

```
program backwrd3
integer nbuffr
parameter ( nbuffr = 25 000 000)
common ibuffr(nbuffr)
call sepcom ( nbuffr )
end

function funcbc ( ichois, x, y, z )
implicit none
integer ichois
double precision funcbc, x, y, z
funcbc = 64d0*(1d0-z)*(z-0.5d0)*y*(1d0-y)
end
```

The iteration process is carried out by starting with the Stokes solution, followed by only Picard iterations. The reason is that Picard in combination with an iterative solver has a better convergence behavior.
Since we are using an iterative solver we must take some precautions.

- The storage method of the large matrix must be set to 6, which means that a compact storage for a non-symmetric matrix is applied.
- We have to use the integrated method, i.e. type 902 or 903.
- Due to the incompressibility condition it is necessary to renumber the unknowns such that first the velocities and then the pressures per level are used. There are three velocity unknowns per point and in the centroid of the element we have 4 pressure unknowns (pressure and gradient of pressure). The velocity physical degrees of freedom have sequence numbers 1, 2 and 3, the pressure physical degrees of freedom have sequence numbers 4, 5, 6 and 7. Hence we use: `renumber levels (1,2,3),(4,5,6,7)` in the problem input.

- The linear solver requires some extra input.
  The preconditioner used is ILU which is the most robust one.
  The linear solver is part of a non-linear iteration process, so that we can start with the solution of the previous non-linear iteration.
  In the linear solver, we are only improving the solution from the previous non-linear iteration and so it suffices to use an accuracy of two extra digits, which means that we set the accuracy equal to 10^{-2}.

In this way we get the following input file:

```
*****************************************************************
*
*      File:  backwrd3.prb
*
*      Backward facing step in R^3
*
*****************************************************************
constants
   vector_names
      velocity
      pressure
end
problem
   # Define type of elements
   types
   elgrp1=902               # Type number for Navier-Stokes, without swirl
                            # 7-point triangle
                            # Approximation 7-point extended triangle
                            # Direct method

   # Define where essential boundary conditions are present

   essbouncond
      surfaces(s3,s4)       # Fixed side walls
      surfaces(s6)          # Lower wall
      surfaces(s8)          # Upper wall
      surfaces(s5)          # Instream boundary

   # Renumber such that per level the velocities are treated before the
   # pressures

   renumber levels (1,2,3),(4,5,6,7)

end

* define type of matrix

matrix
   storage_scheme = compact  # Non-symmetrical compact matrix
                             # So an iterative linear solver will be applied
end

* Create start vector and put the essential boundary conditions into this
* vector
```

```
essential boundary conditions
   surfaces(s5), degfd1, func = 1      # Quadratic inflow profile

end

*  Define coefficients for the first iteration

coefficients
   elgrp1 ( nparm=20 )    # The coefficients are defined by 20 parameters
   coef7  = 1             # 2:  Density
   coef12 = 0.01          # 8:  Value of etha (viscosity)
end

*  Define the coefficients for the next iterations

change coefficients, sequence_number = 1   # Input for iterations 2, 3, ...
   elgrp1
      icoef5 = 1             # 3:  Type of linearization (1=Picard iteration)
end

* Define the parameters for the non-linear solver

nonlinear_equations
   global_options, maxiter=20, accuracy=1d-4,print_level=2, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
end

* Define the parameters for the linear solver

solve
  iteration_method = cg, preconditioning = ilu, print_level=1 //
  start=old_solution, accuracy = 1d-2
end

* Define output, and compute pressure

output
   v1 = icheld=7
end
end_of_sepran_input
```

To run the program the following steps are performed:

```
    seplink backwrd3
    backwrd3 < backwrd3.prb > backwrd.out
```

Finally some post-processing actions are carried out by program SEPPOST using the following input file.

```
*****************************************************************************
*
*      File:  backwrd3.pst
```

```
*
*      Backward facing step in R^3
*
*****************************************************************************
post processing

   # The velocity in the symmetry plane is computed
   # In order to get the components in the plane we need the option
   # transformation=plane_oriented

   compute velocity_in_symmetry_plane = intersection velocity, plane(y=0.5), //
     numbunknowns=3 transformation=plane_oriented

   # The pressure in the symmetry plane is computed

   compute pressure_in_symmetry_plane = intersection pressure, plane(y=0.5)

   # Velocity and pressure in the symmetry plane are plotted

   plot vector velocity_in_symmetry_plane
   plot contour pressure_in_symmetry_plane

end
```

Figure 7.1.4.5 shows the velocity and the pressure in the symmetry plane (y=0.5).

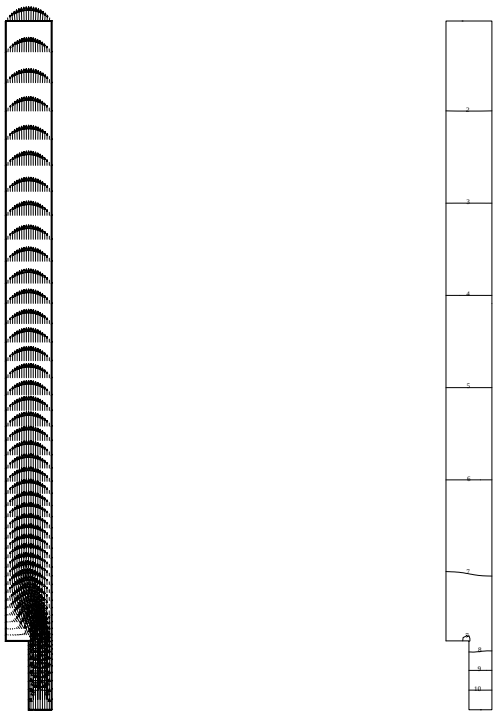


Figure 7.1.4.5: Velocity and pressure in symmetry plane

7.1.5 Time-dependent incompressible flow around a cylinder

In this example we consider vertex shedding behind a circular cylinder as treated by Frans van de Vosse in his thesis (1987). To get this example in your local directory use the command:

```
sepgetex karman
```

To run the example use the commands:

```
sepmesh karman.msh
view the plots
sepcomp karman.prb
seppost karman.pst
view the plots
```

To demonstrate the behaviour of time integration methods, the vortex shedding behind a circular cylinder with diameter $D = 1$ is simulated. The geometry is shown in Figure 7.1.5.1.
At inflow (curves C6 and C10) uniform Dirichlet inflow boundary conditions are used ($u = 1, v =$

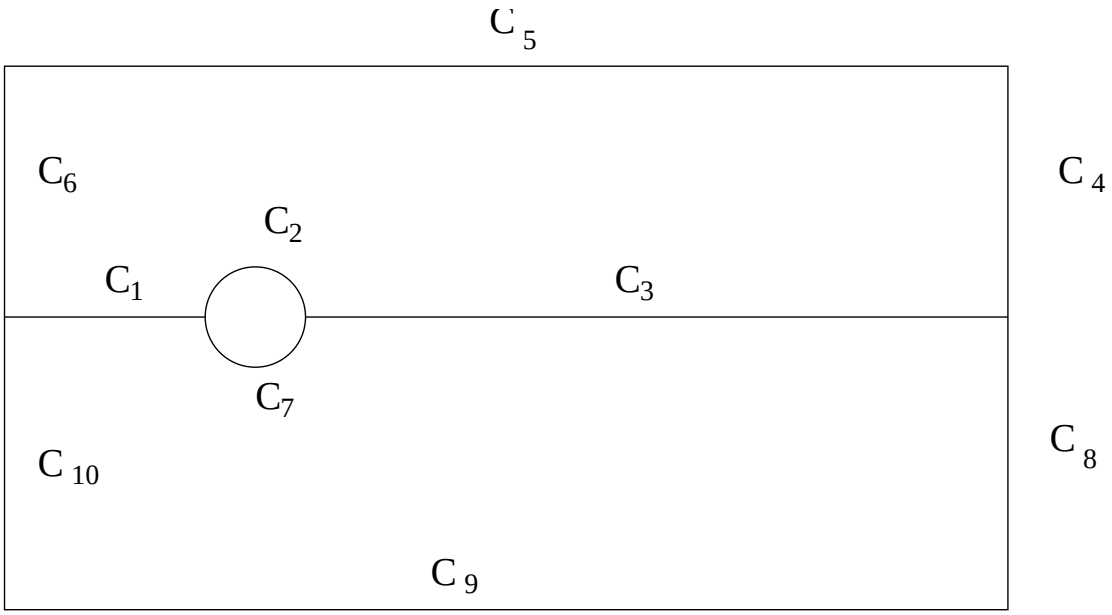


Figure 7.1.5.1: Geometry for vortex shedding problem

0), at outflow (C4 and C8) we assume uniform stress-free boundary conditions. These boundary conditions have the smallest influence on the flow. At the two parallel outer boundaries (C5 and C9) we assume the same given velocity as at the inflow. At the cylinder (curves C2 and C7) a no-slip boundary condition is given.
To create the mesh program SEPMESH is used with the following input file may be used:

```
*  karman.msh
#
#      mesh for vortex shedding problem
#
constants
   reals
      left = -5
      right   = 17
      t = 6
```

```
      r = 0.5
end
mesh2d
   coarse(unit=1)
   points
      p9=(0,0)                # centre of cylinder
      p3=( r,0,0.3)           # point at the right of the cylinder
      p2=(-r,0,0.3)           # point at the left of the cylinder
      p1=( left,0,1)
      p6=( left, t,1)
      p8=( left,- t,1)
      p4=( right,0,1.3)
      p5=( right, t,1.5)
      p7=( right,- t,1.5)
   curves
      c1 = cline2(p1,p2)
      c2 = carc2(p2,p3,-p9)
      c3 = cline2(p3,p4)
      c4 = cline2(p4,p5)
      c5 = cline2(p5,p6)
      c6 = cline2(p6,p1)
      c7 = carc2(p2,p3,p9)
      c8 = rotate c4 (p4,p7)
      c9 = translate c5 (p7,p8)
      c10 = rotate c6 (p8,p1)
   surfaces
      s1 = general4 ( c1,c2,c3,c4,c5,c6 )
      s2 = reflect s1 ( c1,c7,c3,c8,c9,c10 )  # creates a symmetrical mesh
   plot
end
```

The mesh is made completely symmetrical with respect to lower and upper part. This is achieved by the command reflect.
The density of the mesh is defined by the given coarseness. In the neighbourhood of the cylinder the length of the elements is 0.3 times the unit length, this length is taken much larger at the points far away from the cylinder. Figure 7.1.5.2 shows the mesh generated. The boundary conditions in this case are simple. At the inflow and both parallel boundaries we use the uniform velocity.
At the outflow we use the least restrictive outflow boundary conditions, i.e. zero stress.
At the cylinder we use the no-slip condition.
If no precautions are taken both the Euler implicit and Crank Nicolson time integration reach a steady state after about 30 time steps. Due to the symmetry of the mesh and boundary conditions, the vortex shedding was not generated spontaneously. To trigger the vortex shedding, the initial field has been disturbed by setting the velocity of the cylinder equal to 0.1 in y-direction. The boundary conditions at the cylinder are kept at zero.
Following van de Vosse, 10 Euler Implicit steps were performed to damp this distortion and to avoid a too important influence on the flow field. If the Euler implicit scheme is continued the solution again converges to the steady state solution. However, the Crank-Nicolson scheme performs excellent and shows the typical von Karmann vortices one expects. One may try to start with the Crank Nicolson scheme immediately, but since this scheme has no damping properties, the transient will never be damped.
In our example we follow van de Vosse and take the following time steps: $0 \leq t \leq 10,\ \Delta t = 1$ Euler Implicit, followed by Crank Nicolson with $10 \leq t \leq 60,\ \Delta t = 1$, $60 \leq t \leq 75,\ \Delta t = 0.5$ and $75 \leq t \leq 105,\ \Delta t = 0.25$. The results at time $t = 30$ to $t = 105$ with steps 1 are written to the files sepcomp.inf and sepcomp.out for postprocessing purposes.

The corresponding input file is given by

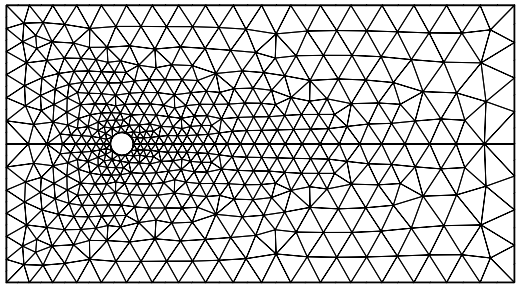


Figure 7.1.5.2: Mesh for vortex shedding problem

```
*  karman.prb
*  problem definition for vortex shedding problem
#
#  Define some general constants
#
set warn off   ! suppress warnings
constants           # See Users Manual Section 1.4
   vector_names
      velocity
end

problem
   types
      elgrp1 = 900                # Standard Navier-Stokes
   essbouncond
      curves(c2)                # part of the cylinder
      curves(c5)                # upper boundary
      curves(c6)                # inflow
      curves(c7)                # other part of the cylinder
      curves(c9)                # lower boundary
      curves(c10)               # inflow
end
*
*  Definition of matrix structure
*
matrix
   # Non-symmetrical profile matrix, So a direct method will be applied
end
*
*  Define initial conditions
*
create vector
```

```
  degfd1 , (value = 1)              # Start vector = (1,0)
  degfd2 , (value = 0)
  degfd2, curves(c2), value = 0.1  # At the cylinder we start with v = 0.1
  degfd2, curves(c7), value = 0.1
end
*
*  Essential boundary conditions
*
essential boundary conditions
   curves(c5), degfd1=value=1                # upper boundary
   curves(c6), degfd1=value=1                # inflow
   curves(c9), degfd1=value=1                # lower boundary
   curves(c10), degfd1=value=1               # inflow
   curves(c2), value = 0                     # cylinder
   curves(c7), value = 0                     # cylinder
end
*
*  Definition of coefficients for the Navier-Stokes equation (t=0 only)
*
coefficients
   elgrp1(nparm=20)
     icoef5  = 2                  # Newton linearization
      coef6  = 1d-6               # penalty parameter eps
      coef7 =  1                  #   rho
      coef12= .01                 #   eta
end
*
* Define the time integration
*
time_integration, sequence_number = 1
   method = theta
   tinit = 0                       # theta method (EI and CN)
   tend  = (10,60,75,105)          # end times of intervals
   tstep = (1,1,0.5,0.25)          # time steps of intervals
   theta=(1,0.5,0.5,0.5)           # corresponding theta values
   toutinit = 30                   # start writing at t=30
   toutend  = 150
   toutstep = 1
   seq_boundary_conditions = 1
   seq_coefficients = 1
   seq_output = 1
   mass_matrix = constant
end
```

In fact it is not necessary to start with Euler implicit and then proceed with Crank Nicolson. It is also possible to use the generalized theta method or the fractional step method. These methods are both accurate and have sufficient damping properties to damp the transient, without damping the vortices. In fact if these method were used the vortex shedding had been reached at an earlier time.

With program seppost it is possible to show the results of the computations. If all time steps are shown a nice movie of the vortex shedding process is produced. However, for the manual we only plot the results at time 30, 55, 80 and 105.
The corresponding input file is given by

```
* File: karman.pst
```

```
*       input for seppost
postprocessing
   compute stream_function = stream function velocity
   time = (0, 150, 25)
      plot vector velocity, factor=.15
      plot contour stream_function
      plot coloured contour stream_function, nlevel=21, mincolour=51
   time history (0,150) plot point(10,0) velocity, degfd=2
end
```

Figures 7.1.5.3 to 7.1.5.6 show the velocity vectors at these time levels. To show the vortices we

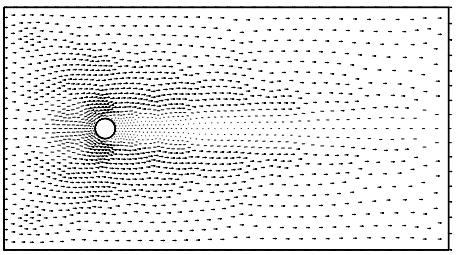


Figure 7.1.5.3: Vector plot of the velocity at t=30

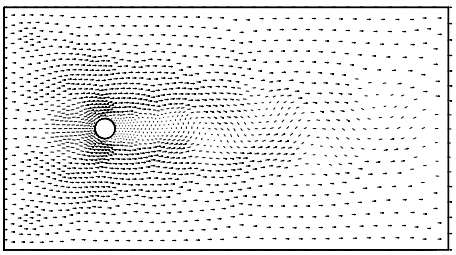


Figure 7.1.5.4: Vector plot of the velocity at t=55

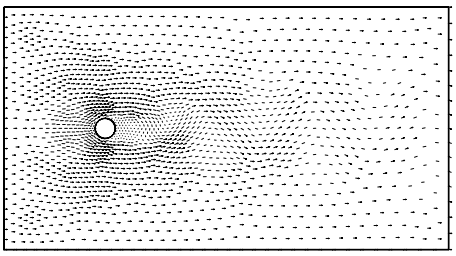


Figure 7.1.5.5: Vector plot of the velocity at t=80

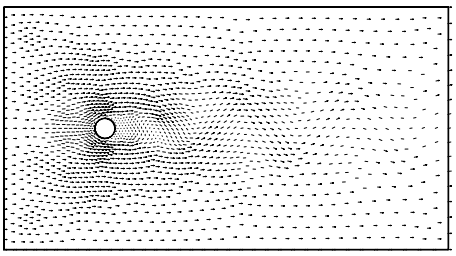


Figure 7.1.5.6: Vector plot of the velocity at t=105

have computed the stream function. Mark that in this time-dependent case the stream lines are not particle trajectories. However, stream lines give a nice insight in the vortex shedding process. Figures 7.1.5.7 to 7.1.5.10 show the stream lines at these time levels.
Figures 7.1.5.11 to 7.1.5.14 show the coloured stream levels at these time levels. Finally in Figure 7.1.5.15 the velocity component in y-direction in point (0,10) is plotted as function of time. From the fluctuations the Strouhal number can be detected. See van de Vosse for the details.

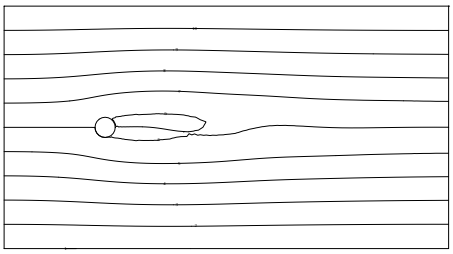


Figure 7.1.5.7: Stream lines at t=30

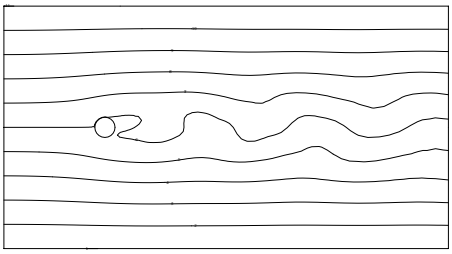


Figure 7.1.5.8: Stream lines at t=55

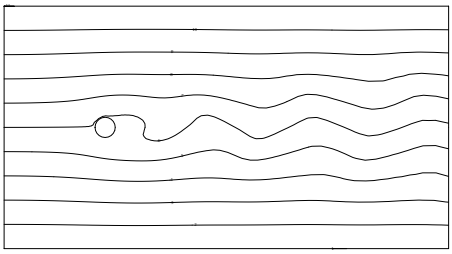


Figure 7.1.5.9: Stream lines at t=80

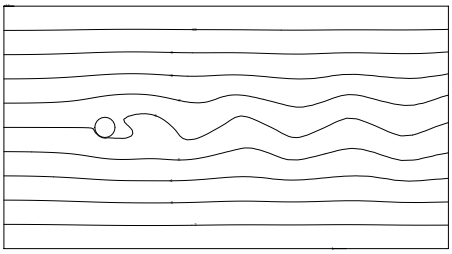


Figure 7.1.5.10: Stream lines at t=105

7.1.6 Free Surface Flow; co-flowing streams

In this example we consider the laminar flow out of two parallel channels that come together. See Figure 7.1.6.1 for a definition of the geometry. The driving forces of the flow are pressure differences. At the outflow (curves C_3–C_4) the pressure is assumed to be zero. At the inflow part of the channels, the pressure levels are different; $p = 2$ at C_7, and $p = 1$ at C_8. The curve C_9 is a solid wall that divides the co-flowing streams. The curve C_{10} is the initial position of the streamline between the two co-flowing streams. The position of this streamline must be determined during the calculations. To get this example into your local directory use:

```
sepgetex coflow
```

and to run it use:

```
sepfree coflow.prb
seppost coflow.pst
```

The initial mesh has also been given in Figure 7.1.6.1.

The equations to be solved are the incompressible Navier-Stokes equations.
The boundary conditions can be formulated as:

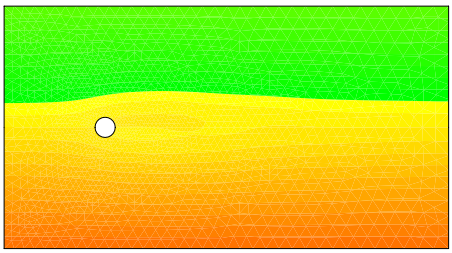


Figure 7.1.5.11: Stream levels at t=30

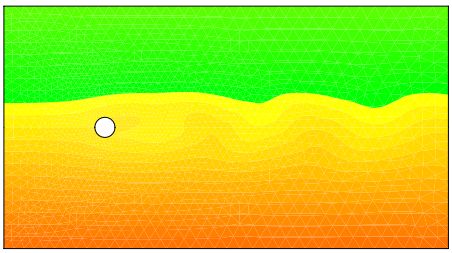


Figure 7.1.5.12: Stream levels at t=55

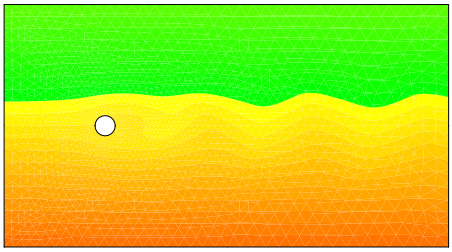


Figure 7.1.5.13: Stream levels at t=80

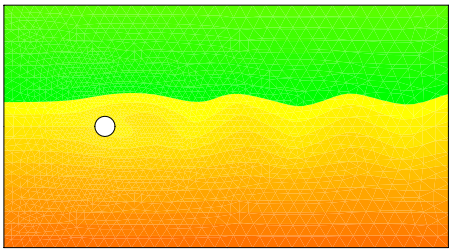


Figure 7.1.5.14: Stream levels at t=105

- $\mathbf{v} = 0$ at fixed walls: C_1, C_2, C_9.
- Symmetry conditions ($v_n = 0,\ \sigma_t = 0$) at C_5–C_6.
- Pressure level uniform at outflow ($\sigma_n = -p = 0$) at C_3–C_4, and fully-developed flow, i.e. $v_t = 0$. It is, however, *essential* that v_t is *not* prescribed at the last point of C_3 and the first point of C_4.
- Pressure level uniform at inflow ($\sigma_n = -p = -2$ at C_7; $\sigma_n = -p = -1$ at C_8), and fully-developed flow i.e. $v_t = 0$.
- The streamline C_{10} is not known, hence this is a so-called free boundary. In order to determine the position of this streamline an extra boundary condition is necessary. The standard boundary conditions is of course that the velocity is continuous, i.e. the velocity at the streamline belongs to both regions. Furthermore, along a streamline we have $v_n = 0$. This condition is used to compute the free boundary C_{10} during the iterations.

It is not required to compute a boundary integral explicitly along a curve when is it zero everywhere. This is the case when $\sigma_n = 0,\ v_t = 0$ or the combination $\sigma_t = 0,\ v_n = 0$. So a boundary integral is needed only at C_7 and C_8.

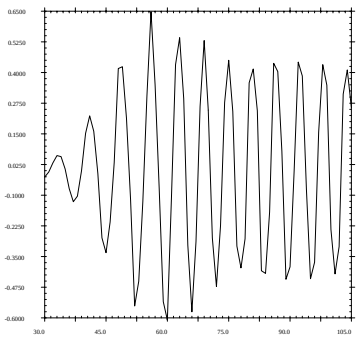


Figure 7.1.5.15: Velocity component in y-direction as function of time

The properties of the fluid have been chosen equal to each other, and $\rho = 1,\ \eta = 1$. This prevents instabilities due to property differences. Extended quadratic triangles, in combination with the penalty function method have been applied. For the internal elements type number 900 has been used, the boundary integrals are computed by boundary elements of type number 910.

The unknown free boundary is adapted using the so-called film method of Caswell and Viriyayuthakorn (1983). Starting from an initial guess the Navier-Stokes equations are solved using the boundary conditions given above. At the common streamline only the trivial continuity boundary conditions are applied. After each solution of the equations the free boundary is adapted to the third boundary condition, until the difference between the computed velocity in two succeeding iterations is small enough.

Program SEPFREE does the mesh generation, solves the problem, adapts the mesh, solves again, until convergence has been reached.

The structure of the main program is defined by the user. To that end the block "STRUCTURE" is used. Three vectors are defined:

1. the velocity vector **v**
2. the pressure p
3. the stress tensor **t**

The structure of the program is as follows:

- First the initial mesh is generated and the problem description is read. This is the standard start of program SEPFREE.
- Next the essential boundary conditions are prescribed at $t = 0$.
- Finally the free boundary problem is solved.
  In the first step the linear Stokes equations are solved.
  In all other iterations the convective terms are linearized by Picard. To that end the coefficients are changed before the free boundary loop.
  It is not necessary to solve the non-linear equations in each step. In fact one iteration (i.e.

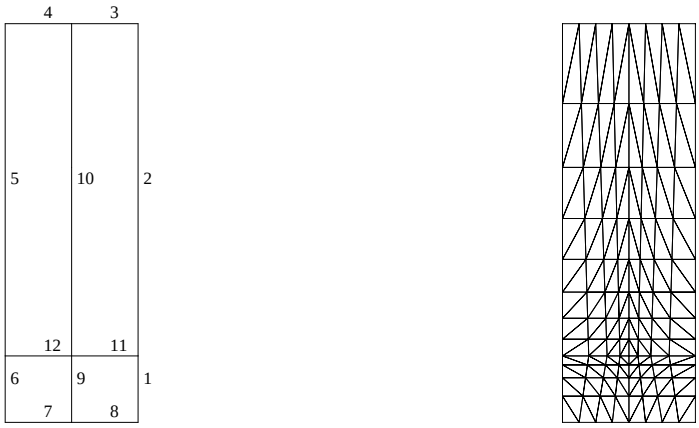


Figure 7.1.6.1: Geometry definition for the co-flowing streams problem; initial mesh

solution of a linear system) is sufficient, since after each iteration the boundary is adapted. Since this process is a Picard iteration itself, it makes no sense to use a Newton linearization of the convective terms.

- Once the process has been converged, the pressure and the stress are computed and all vectors are written to the files `sepcomp.out` and `sepcomp.inf` for post-processing purposes.

The following input file has been used:

```
*coflow.prb
constants
   vector_names
      velocity
      pressure
      stress
end
mesh2d
   points
      p1=(2,0)
      p2=(2,1)
      p3=(2,6)
      p4=(1,6)
      p5=(0,6)
      p6=(0,1)
      p7=(0,0)
      p8=(1,0)
      p9=(1,1)
   curves
      c1 =line2(p1,p2,nelm=4,ratio=2,factor=0.7)
      c2 =line2(p2,p3,nelm=8,ratio=4,factor=0.8)
      c3 =line2(p3,p4,nelm=4)
      c4 =line2(p4,p5,nelm=4)
      c5 =line2(p5,p6,nelm=8,ratio=2,factor=0.8)
```

```
      c6 =line2(p6,p7,nelm=4,ratio=4,factor=0.7)
      c7 =line2(p7,p8,nelm=4)
      c8 =line2(p8,p1,nelm=4)
      c9 =line2(p8,p9,nelm=4,ratio=2,factor=0.7)
      c10=line2(p9,p4,nelm=8,ratio=4,factor=0.8)
      c11=line2(p2,p9,nelm=4,ratio=2,factor=0.7)
      c12=line2(p9,p6,nelm=4,ratio=4,factor=0.7)

   surfaces
      s1=rectangle4(c1,c11,-c9,c8)
      s2=rectangle4(c2,c3,-c10,-c11)

      s3=rectangle4(c9,c12,c6,c7)
      s4=rectangle4(c10,c4,c5,-c12)
   plot(jmark=5, numsub=4,plotfm=15)
end
problem
   types
      elgrp1=(type=900)
   natboundcond
      bngrp1=(type=910)
      bngrp2=(type=910)
   bounelements
      belm1=curves(shape=2,c8)
      belm2=curves(shape=2,c7)
   essbouncond
*     symmetry
      degfd1=curves (c5,c6)

*     fixed wall
      degfd1,degfd2=curves (c1,c2)
      degfd1,degfd2=curves (c9)

*     outlet
      degfd1=curves200(c3)
      degfd1=curves100(c4)

*     inlet
      degfd1=curves (c7,c8)
end
coefficients
   elgrp1 (nparm=20)
     icoef2 = 1        # Newtonian fluid
     icoef5 = 0        # stokes
      coef6 = 1d-8     # penalty parameter
      coef7 = 1        # rho
      coef12= 1        # etha
   bngrp1 (nparm=15)
     icoef1 = 1        # normal and tangential direction
      coef6 = -1       # Pressure boundary condition
   bngrp2 (nparm=15)
     icoef1 = 1        # normal and tangential direction
      coef6 = -2       # Pressure boundary condition
end
```

```
change coefficients
   elgrp1
     icoef5 = 1        # Picard
end

adapt_boundary
   curves=c10, adaptation_method=film_method, quadratic, exclude_begin = both
   exclude_end = second
end

adapt_mesh
end

structure
   prescribe_boundary_conditions, velocity
   solve_linear_system
   change_coefficients
   start_stationary_free_boundary
      solve_linear_system
   end_stationary_free_boundary
   derivatives, seq_deriv=1, pressure
   derivatives, seq_deriv=2, stress
   output
end
stationary_free_boundary
   maxiter=20, miniter=3, print_level=2, accuracy=1d-6, criterion = relative
end
derivatives, sequence_number=1
   icheld = 7                    # pressure
end
derivatives, sequence_number=2
   icheld = 6                    # stress
end
end_of_sepran_input
```

It is essential that the velocity v_t in the last point of C_3, which is the same as the first point of C_4, is *not* prescribed as an essential boundary condition. The adaptation of the position of curve C_{10} is subjected to the following constraints:

- The first point of C_{10} has a *fixed* position.
- The last point of C_{10} has a *fixed* x_2 co-ordinate.

The resulting mesh has been plotted in Figure 7.1.6.2.

The commands that are required for the program SEPPOST are given below:

```
*coflow.pst

post processing

*  Print all three vectors completely

   print velocity
   print pressure
   print stress
```

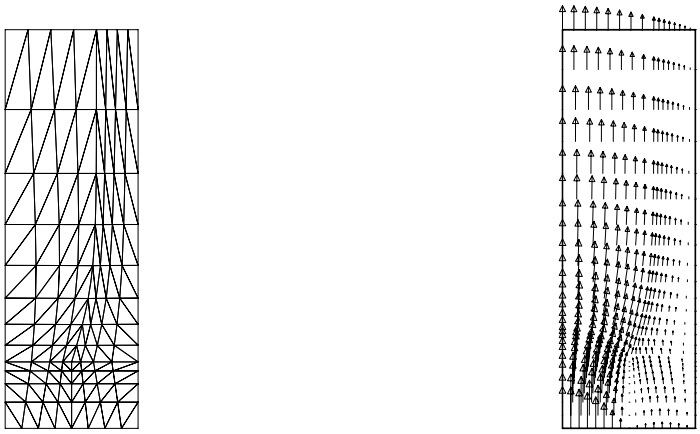


Figure 7.1.6.2: Final mesh and velocity vectors

```
*  Compute stream function, store in stress, and name this vector

   compute stream_function = stream function velocity

* Plot the results

   plot mesh
   plot vector velocity                 # Vector plot of velocity
   plot contour pressure (nlevel=25)    # Contour plot of pressure
   plot contour stream_function (nlevel=20)   # Contour plot of stream function

end
```

The resulting velocity vectors have also been plotted in Figure 7.1.6.2. The pressure contour lines (isobars) and the streamlines have been plotted in Figure 7.1.6.3.

Figure 7.1.6.3: Isobars and streamlines

7.1.7 Convection in the earth mantle

Studying convection in planetary interiors requires a -costly- solution of the 3D Stokes and heat equations in spherical geometry. A reduction in computational cost can be made by approximating the sphere by a 2D cylinder geometry. For convection in the silicate mantle of Earth the geometry shown in Figure 7.1.7.1a may be used. Gravity is directed towards the centre of the planet and the

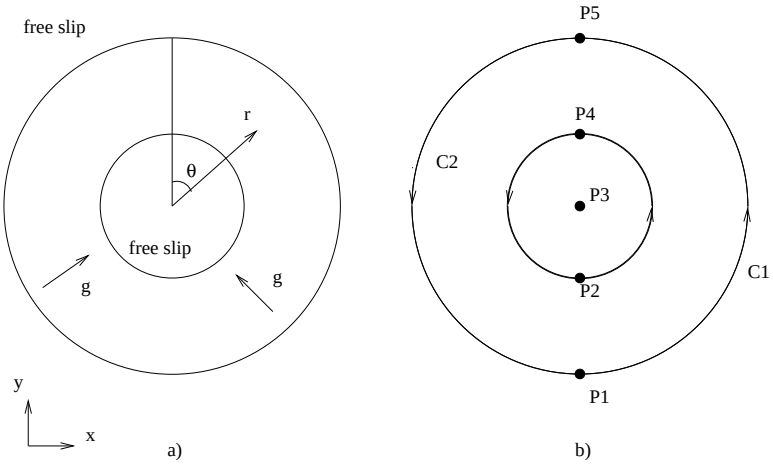


Figure 7.1.7.1: Definition of region and boundary conditions

boundary conditions at top and bottom of the mantle are free-slip. The Stokes equations may be solved in Cartesian coordinates, but this requires local transformations for the free-slip boundary conditions. The reason is that we want to prescribe the normal component ($u_n = 0$) of the velocity, but that the tangential component is free. This component is not in the direction of the co-ordinate axis.
Since the solution is fixed upon an additive constant, it is necessary to prescribe the velocity in one point.

To get this example into your local directory use:

```
sepgetex earth
```

and to run it use:

```
sepmesh earth.msh
seplink earth
earth < earth.prb
seppost earth.pst
```

Below is a simple example that show how the mesh needs to be defined to make sure that the local transformations work correctly.

The mesh can be defined by the five points and four curves shown (with orientation) in Figure 7.1.7.1b. For the local transformations the inner and outer boundary are defined as two separate curves (C5 and C6), where the orientation of the inner curve is reversed, such that the normal to the curve points away from the computational domain.

In the computational part the boundary conditions are prescribed on curves C5 and C6. The isoviscous, incompressible Stokes equations are solved using the penalty function method. The gravity vector is directed to the centre of the cylinder and consequently the buoyancy forces are described by two components:

$$\mathbf{f} = -f \begin{bmatrix} sin(\theta) \\ cos(\theta) \end{bmatrix}, \tag{7.1.7.1}$$

where θ is the co-latitude. In the example below, the buoyancy forces are described by a simple harmonic perturbation in θ. The applied buoyancy force leads to a pattern of eight convection cells, symmetric around x=0 and y=0. The outer curves for the local transformation have been chosen such that a counter clockwise direction is used. In this specific example this is not necessary.

The following input file may be used to define the mesh:

```
*earth.msh
mesh2d
  coarse(unit=20)
  points
    p1=(0, -1.0, 0.005)
    p2=(0, -0.5, 0.005)
    p3=(0,  0.0, 0.005)
    p4=(0,  0.5, 0.005)
    p5=(0,  1.0,0.005)
  curves
    c1  = carc2(p1,p5,p3)
    c2  = carc2(p5,p1,p3)
    c3  = carc2(p2,p4,p3)
    c4  = carc2(p4,p2,p3)
    c5  = curves(c3,c4)
    c6  = curves(c1,c2)
    c7  = cline2(p1,p2)
  surfaces
    s1 =general4(c1,c2,c7,-c4,-c3,-c7)
  plot
end
```

The mesh created can be found in Figure 7.1.7.2.

Since the right-hand side is a function of the co-ordinates, it is necessary to write a simple program provided with the function subroutine FUNCCF. This program is given below:

```
     program earthconvection
      call sepcom(0)
      end
! ************************************************************
!     FUNCCF
!
!     Define buoyancy forces to drive flow in a cylindrical
!     geometry. It is assumed that the gravity points to the
!     center of the cylinder (as if to model a self-gravitating
!     planet): g = - (sin(theta),cos(theta))
!
!     The buoyancy forces specified below should lead to an 8-cell
!     convection pattern, symmetrical around x=0.
! ************************************************************
      function funccf(ifunc,x,y,z)
      implicit none
      integer ifunc
      double precision funccf,x,y,z
      double precision r,theta,sint,cost,asin,pi
      parameter(pi=3.1415926d0)

!     --- find polar coordinates for this point

      r = sqrt(x*x+y*y)
      theta = asin(y/r)

!     --- sin(theta),cos(theta)

      sint = x/r
      cost = y/r

      if ( ifunc.eq.1 ) then
         funccf = cos((theta+pi/2d0)*4d0)  * sint
      else if ( ifunc.eq.2 ) then
         funccf = cos((theta+pi/2d0)*4d0)  * cost
      end if

      end
```

The corresponding input file is a standard function for program SEPCOMP. It has the following shape:

```
*earth.prb
constants           # See Users Manual Section 1.4
   vector_names
      velocity
end
problem
   types
      elgrp1=(type=900)
   essboundcond
      degfd2=points(p1)
      degfd1=curves(c5)
      degfd1=curves(c6)
   localtransform
```

```
      degfd1,degfd2=curves(-c5)
      degfd1,degfd2=curves(c6)
end
matrix
   symmetric
end

coefficients
   elgrp1 (nparm=20)
      icoef2 = 1
      coef6 = (value=1d-6)
      coef7 = (value=1)
      coef9 = (func=1)
      coef10 = (func=2)
      coef12 = (value=1)
end

solve
   positive_definite
   direct_solver = profile
end
```

The output of this program may be vizualised with program seppost in combination with sepview. In the input file below we plot the velocity vector

```
*earth.pst
postprocessing
plot vector velocity
end
```

The velocity field is plotted in Figure 7.1.7.2.

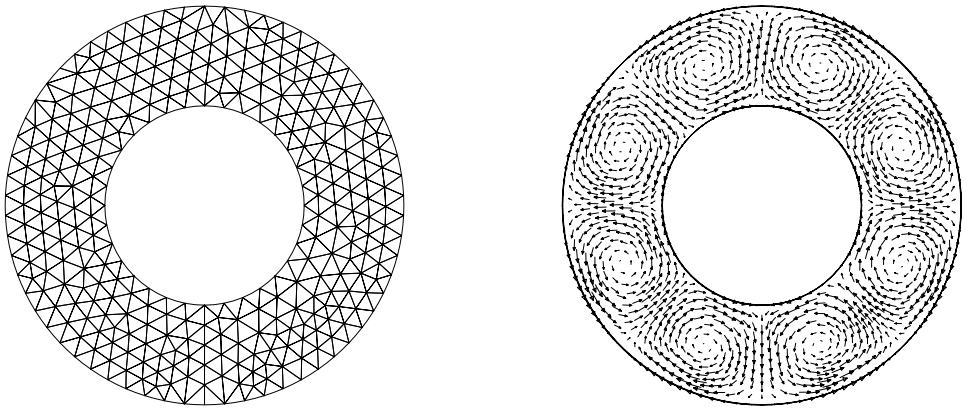


Figure 7.1.7.2: Mesh and velocity vectors

7.1.8 Application of all 2D elements to a simple channel flow

In this section we consider a simple channel flow (Cartesian co-ordinates) for low Reynolds numbers. The exact solution is a quadratic velocity profile perpendicular to the flow direction and a linear pressure field. The reason to solve this simple problem is that it shows how the various element shapes and element types may be used to solve the same problem.
In order to get these examples into your local directory use the command

```
sepgetex channelxx
```

where xx is a 2 digit number. The following numbers are available:

number	shape	type	description
11	4	900	extended quadratic triangle, penalty method
12	5	900	linear quadrilateral, penalty method
13	6	900	biquadratic quadrilateral, penalty method
21	6	902	biquadratic quadrilateral, integrated method
22	7	902	extended quadratic triangle, integrated method
23	9	902	bilinear quadrilateral, integrated method
31	7	901	extended quadratic triangle, integrated method (elimination)
41	3	903	linear triangle, Taylor Hood
42	4	903	quadratic triangle, Taylor Hood
43	6	903	biquadratic quadrilateral, Taylor Hood
44	10	903	extended linear triangle, Taylor Hood

Figure 7.1.8.1 shows the channel and the corresponding curves. In curve C4 we have a parabolic

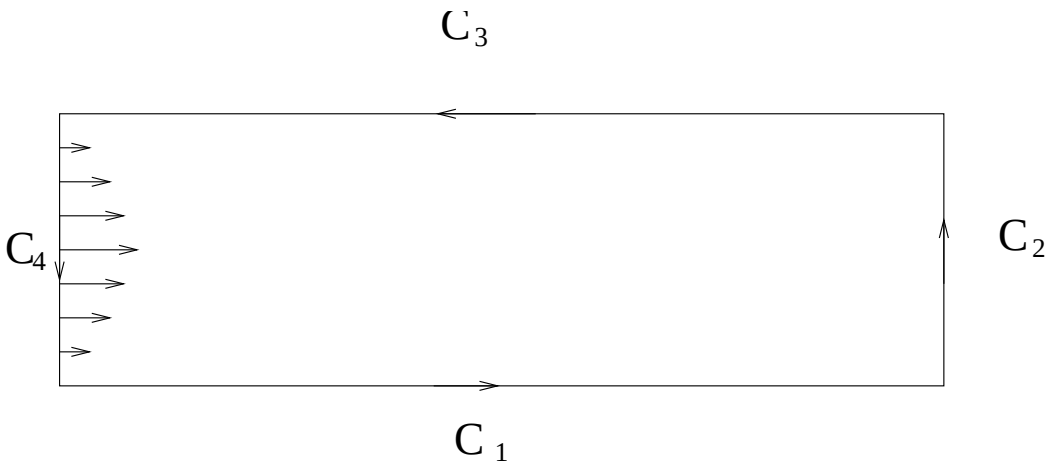


Figure 7.1.8.1: Definition of region and boundary conditions

inflow profile. This means that the tangential velocity is 0 and the normal velocity component is prescribed by a quadratic function.
The curves C1 and C3 denote fixed walls and at curve C2 we prescribe parallel outflow. In all our examples we use a 8×8 linear or 8×8 quadratic subdivision in elements.

The exact solution is shown in Figures 7.1.8.2 (velocity vectors), 7.1.8.3 (isobars), 7.1.8.4 (colored pressure levels), 7.1.8.5 (stream lines) and 7.1.8.6 (colored stream function levels).

We consider the input of the different methods separately.

Penalty function approach For these elements type number 900 must be used. A penalty function parameter must be chosen, which for scaled problems is usually of the order 10^{-6}. The quadratic velocity profile is prescribed with the option QUADRATIC, and since we take a maximum velocity of 1, MAX does not have to be given.

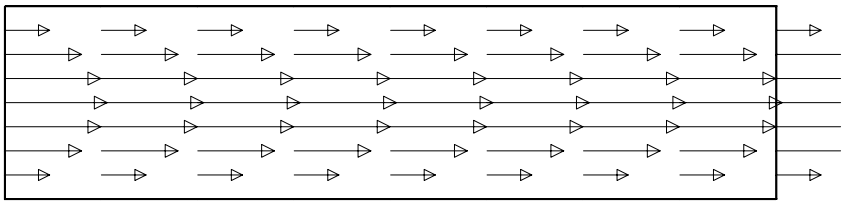


Figure 7.1.8.2: Vector plot of velocity field

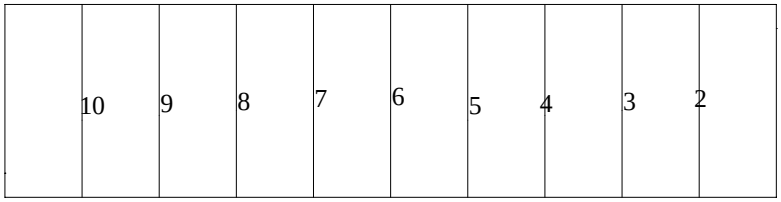


Figure 7.1.8.3: Isobars

To show how one can compute special quantities during the computation, a structure block is provided. In this block not only the velocity is computed, but also the pressure and a boundary integral of the pressure over the inflow curve C4 is computed and printed.
At this moment 3 different element shapes are available for the 2D case.

shape = 4 The input for program SEPMESH is given in the following input file (channel11.msh):

```
#  channel11.msh
#
#  mesh file for 2d channel problem
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepmesh channel11.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the channel
```

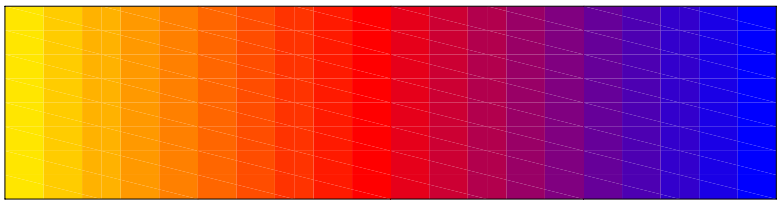


Figure 7.1.8.4: colored pressure levels

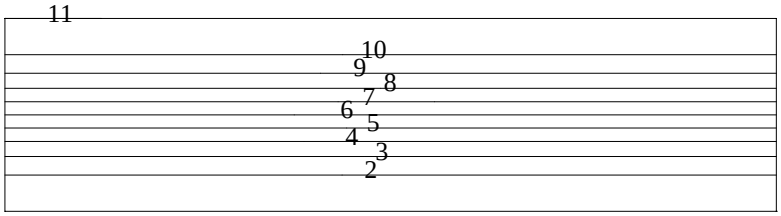


Figure 7.1.8.5: stream lines

```
      length = 4              # length of the channel
   integers
      n = 4                   # number of elements in length direction
      m = 4                   # number of elements in width direction
      shape_cur = 2           # Type of elements along curves
                              # quadratic elements
      shape_sur = 6           # Type of elements in surface
                              # Quadratic triangles
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)              # Left under point
      p2=(length,0)         # Right under point
      p3=(length,width)     # Right upper point
      p4=(0,width)          # Left upper point
#
```

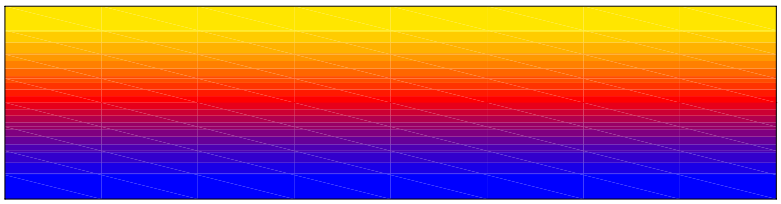


Figure 7.1.8.6: colored stream function levels

```
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line shape_cur (p1,p2,nelm=n)     # lower wall
      c2=line shape_cur (p2,p3,nelm=m)     # outflow boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper wall
      c4=line shape_cur (p4,p1,nelm=m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle shape_sur (c1,c2,c3,c4)

   plot                            # make a plot of the mesh
                                   # See Users Manual Section 2.2

end
```

The input file for SEPCOMP is given by the file channel11.prb:

```
# channel11.prb
#
#  problem file for 2d channel problem
#  penalty function approach
#  problem is stationary and non-linear
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepcomp channel11.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
```

```
      eps       = 1d-6              # penalty parameter for Navier-Stokes
      rho       = 1                 # density
      eta       = 0.01              # viscosity
   vector_names
      velocity
      pressure
   variables
      pressure_int
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=900                # Type number for Navier-Stokes, without swirl
                                # See Standard problems Section 7.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      curves(c1)                # Fixed under wall
      curves(c3)                # Fixed side walls and instream boundary
      curves(c4)                # inflow
      degfd2=curves(c2)         # Outstream boundary (v-component given)
                                # All not prescribed boundary conditions
                                # satisfy corresponding stress is zero
end
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because the integral of the pressure over the boundary
#  is required
#
structure                    # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions velocity
   solve_nonlinear_system velocity
 # Compute the pressure
   derivatives pressure
 # Compute the integral of the pressure over curve c2 (outflow boundary)
   boundary_integral,pressure, scalar1 = pressure_int
   print pressure_int, text = 'integral of pressure over curve c2'
 # Write the results to a file
   output
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c4), degfd1, quadratic   # The u-component of the velocity at
                                   # instream is quadratic
                                   # The rest of the vector is 0
```

```
end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef6  = eps        # 6:  Penalty function parameter eps
      coef7  = rho        # 7:  Density
      coef12 = eta        #12:  Value of eta (viscosity)
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end
#
#  Define information with respect to the boundary integral to be computed
#  See Users Manual, Section 3.2.14
#
boundary_integral, sequence_number = 1
   ichint = 1                # Standard integration
   curves = c4               # integral over curve c4
end

# compute pressure
# See Users Manual, Section 3.2.11

derivatives, sequence_number = 1
   icheld=7          # icheld=7, pressure in nodes
                     # See Standard problems Section 7.1
```

```
end

end_of_sepran_input
```

The standard nonlinear algorithm, i.e. start with Stokes, do one step Picard and finally use Newton is applied. However, for this particular problem the solution is reached in two steps due to the fact that the convective terms do not play a role.
The solution with this element is of course exact up to an accuracy of the order of 10^{-6}, which is the penalty function parameter.
The postprocessing input file channel11.pst, which produces the pictures shown before is defined by:

```
#  channel11.pst
#  Input file for postprocessing for channel problem
#  See Manual Standard Elements Section 7.1.8
#
#
#  To run this file use:
#     seppost channel11.pst > channel11.out
#
#  Reads the files meshoutput and sepcomp.out
#
#
postprocessing                    # See Users Manual Section 5.2

#
#  compute the stream function
#  See Users Manual Section 5.2
#  store in stream_function

   compute stream_function = stream function velocity

#  Plot the results
#  See Users Manual Section 5.4

   plot vector velocity              # Vector plot of velocity
   plot contour pressure             # Contour plot of pressure
   plot coloured contour pressure
   plot contour stream_function      # Contour plot of stream function
   plot coloured contour stream_function

end
```

shape = 5 In this case we use an element that does not satisfy the Brezzi Babuska condition. However, still the results are reasonable, due to the fact that at outflow no velocity is prescribed.
One can not expect exact results since the pressure approximation is only constant per element and the velocity approximation is only linear.
The mesh input file channel12.msh is given by:

```
#  channel12.msh
#
#  mesh file for 2d channel problem
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepmesh channel12.msh
```

```
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the channel
      length = 4             # length of the channel
   integers
      n = 8                  # number of elements in length direction
      m = 8                  # number of elements in width direction
      shape_cur = 1          # Type of elements along curves
                             # linear elements
      shape_sur = 5          # Type of elements in surface
                             # Bilinear quadrilaterals
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)               # Left under point
      p2=(length,0)          # Right under point
      p3=(length,width)      # Right upper point
      p4=(0,width)           # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line shape_cur (p1,p2,nelm=n)     # lower wall
      c2=line shape_cur (p2,p3,nelm=m)     # outflow boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper wall
      c4=line shape_cur (p4,p1,nelm=m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle shape_sur (c1,c2,c3,c4)

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2

end
```

The problem file and the postprocessing file are completely identical to the one used for shape = 4.
The pictures for the velocity and the stream lines do not show any difference with the exact solution. The isobars in Figure 7.1.8.7 however, show that the solution is not exact.

shape = 6 In this biquadratic case the solution is again nearly exact. The problem file and

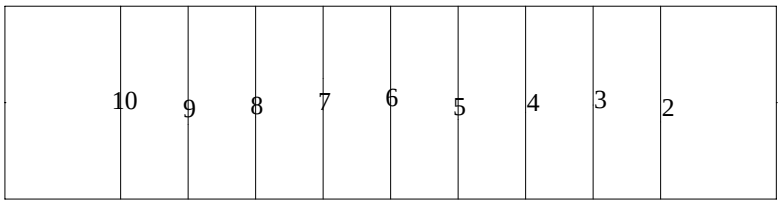


Figure 7.1.8.7: Isobars (shape=5)

the postprocessing file are completely identical to the one used for shape = 4.
The mesh input file channel13.msh is given by

```
#  channel13.msh
#
#  mesh file for 2d channel problem
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepmesh channel13.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the channel
      length = 4             # length of the channel
   integers
      n = 4                  # number of elements in length direction
      m = 4                  # number of elements in width direction
      shape_cur = 2          # Type of elements along curves
                             # quadratic elements
      shape_sur = 6          # Type of elements in surface
                             # Bi-quadratic quadrilaterals
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)              # Left under point
      p2=(length,0)         # Right under point
      p3=(length,width)     # Right upper point
      p4=(0,width)          # Left upper point
```

```
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line shape_cur (p1,p2,nelm=n)     # lower wall
      c2=line shape_cur (p2,p3,nelm=m)     # outflow boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper wall
      c4=line shape_cur (p4,p1,nelm=m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle shape_sur (c1,c2,c3,c4)

   plot                            # make a plot of the mesh
                                   # See Users Manual Section 2.2

end
```

Integrated method In the integrated method, there is no need to prescribe a penalty parameter. However, in this case we must be careful with respect to the solution method since the continuity equation does not contain the pressure. As a consequence the equations corresponding to the pressure unknowns contain a zero at the main diagonal. Since the linear solver does not apply a kind of pivoting it is necessary to order the unknowns such that the first rows of the matrix correspond to velocity unknowns and that rows corresponding to pressure unknowns follow these velocity rows. This can be achieved by the option renumber in the problem file. However, if we start with all velocity unknowns and then all pressure unknowns the size of the matrix is very large. For that reason the option renumber levels is used. If this option is used it is best to take care of a good numbering of the nodes. It is best to start the renumbering with the outflow boundary, since there only a part of the velocity unknowns are prescribed. Furthermore for this problem the Cuthill-McKee numbering is preferred above the standard renumbering. In order to force such a numbering we use the option

```
renumber, start = c2, Cuthill_McKee, always
```

in the mesh input files.
Next we consider the three shapes that are available for type number 902.

shape = 6 The mesh input file is given by:

```
#  channel21.msh
#
#  mesh file for 2d channel problem
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepmesh channel21.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the channel
      length = 4             # length of the channel
```

```
   integers
      n = 4                  # number of elements in length direction
      m = 4                  # number of elements in width direction
      shape_cur = 2          # Type of elements along curves
                             # quadratic elements
      shape_sur = 6          # Type of elements in surface
                             # Bi-quadratic quadrilaterals
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)              # Left under point
      p2=(length,0)         # Right under point
      p3=(length,width)     # Right upper point
      p4=(0,width)          # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line shape_cur (p1,p2,nelm=n)     # lower wall
      c2=line shape_cur (p2,p3,nelm=m)     # outflow boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper wall
      c4=line shape_cur (p4,p1,nelm=m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle shape_sur (c1,c2,c3,c4)

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2
   renumber, start = c2, Cuthill_McKee, always
                                  # Force a renumbering
                                  # See Users Manual Section 2.2

end
```

The problem input file is given by:

```
# channel21.prb
#
#  problem file for 2d channel problem
#  integrated method
#  problem is stationary and non-linear
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepcomp channel21.prb
#
#  Reads the file meshoutput
```

```
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
   vector_names
      velocity
      pressure
   variables
      pressure_int
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=902                # Type number for Navier-Stokes, without swirl
                                # integrated approach
                                # See Standard problems Section 7.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      curves(c1)                # Fixed under wall
      curves(c3)                # Fixed side walls and instream boundary
      curves(c4)                # inflow
      degfd2=curves(c2)         # Outstream boundary (v-component given)
                                # All not prescribed boundary conditions
                                # satisfy corresponding stress is zero
   renumber levels (1,2),(3,4,5)  # renumber the unknowns such that for each
                                # level first we have all velocities and then
                                # all pressures, thus avoiding zero pivots
end
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because the integral of the pressure over the boundary
#  is required
#
structure                   # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions, velocity
   solve_nonlinear_system, velocity
 # Compute the pressure
   derivatives, pressure
 # Compute the integral of the pressure over curve c2 (outflow boundary)
   boundary_integral, pressure, scalar1 = pressure_int
   print pressure_int, text = 'integral of pressure over curve c2'
 # Write the results to a file
   output
```

```
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c4), degfd1, quadratic   # The u-component of the velocity at
                                   # instream is quadratic
                                   # The rest of the vector is 0

end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef7  = rho        # 7:  Density
      coef12 = eta        #12:  Value of eta (viscosity)
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end
#
#  Define information with respect to the boundary integral to be computed
#  See Users Manual, Section 3.2.14
#
boundary_integral, sequence_number = 1
```

```
   ichint = 1                # Standard integration
   curves = c4               # integral over curve c4
end

# compute pressure
# See Users Manual, Section 3.2.11

derivatives, sequence_number = 1
   icheld=7          # icheld=7, pressure in nodes
                     # See Standard problems Section 7.1
end

end_of_sepran_input
```

The input file for the postprocessing is the same as for the penalty function approach.

shape = 7 The mesh input file is given by:

```
#  channel22.msh
#
#  mesh file for 2d channel problem
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepmesh channel22.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the channel
      length = 4             # length of the channel
   integers
      n = 4                  # number of elements in length direction
      m = 4                  # number of elements in width direction
      shape_cur = 2          # Type of elements along curves
                             # quadratic elements
      shape_sur = 7          # Type of elements in surface
                             # Extended quadratic triangles
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)              # Left under point
      p2=(length,0)         # Right under point
      p3=(length,width)     # Right upper point
      p4=(0,width)          # Left upper point
#
#  curves
#
```

```
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line shape_cur (p1,p2,nelm=n)     # lower wall
      c2=line shape_cur (p2,p3,nelm=m)     # outflow boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper wall
      c4=line shape_cur (p4,p1,nelm=m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle shape_sur (c1,c2,c3,c4)

   plot                            # make a plot of the mesh
                                   # See Users Manual Section 2.2
   renumber, start = c2, Cuthill_McKee, always
                                   # Force a renumbering
                                   # See Users Manual Section 2.2

end
```

The input files for SEPCOMP and SEPPOST are the same as for shape 6.

shape = 9 The mesh input file is given by:

```
#  channel23.msh
#
#  mesh file for 2d channel problem
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepmesh channel23.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the channel
      length = 4             # length of the channel
   integers
      n = 8                  # number of elements in length direction
      m = 8                  # number of elements in width direction
      shape_cur = 1          # Type of elements along curves
                             # linear elements
      shape_sur = 9          # Type of elements in surface
                             # Extended bi-linear quadrilaterals
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)              # Left under point
```

```
      p2=(length,0)         # Right under point
      p3=(length,width)     # Right upper point
      p4=(0,width)          # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line shape_cur (p1,p2,nelm=n)     # lower wall
      c2=line shape_cur (p2,p3,nelm=m)     # outflow boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper wall
      c4=line shape_cur (p4,p1,nelm=m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle shape_sur (c1,c2,c3,c4)

   plot                             # make a plot of the mesh
                                    # See Users Manual Section 2.2
   renumber, start = c2, Cuthill_McKee, always
                                    # Force a renumbering
                                    # See Users Manual Section 2.2

end
```

The corresponding problem input file is:

```
# channel23.prb
#
#  problem file for 2d channel problem
#  integrated method
#  problem is stationary and non-linear
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepcomp channel23.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
   vector_names
      velocity
      pressure
   variables
      pressure_int
end
#
```

```
#  Define the type of problem to be solved
#
problem                    # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=902                # Type number for Navier-Stokes, without swirl
                                # integrated approach
                                # See Standard problems Section 7.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      curves(c1)                # Fixed under wall
      curves(c3)                # Fixed side walls and instream boundary
      curves(c4)                # inflow
      degfd2=curves(c2)         # Outstream boundary (v-component given)
                                # All not prescribed boundary conditions
                                # satisfy corresponding stress is zero
   renumber levels (1,2),(3)    # renumber the unknowns such that for each
                                # level first we have all velocities and then
                                # all pressures, thus avoiding zero pivots
end
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because the integral of the pressure over the boundary
#  is required
#
structure                     # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions, velocity
   solve_nonlinear_system, velocity
 # Compute the pressure
   derivatives, pressure
 # Compute the integral of the pressure over curve c2 (outflow boundary)
   boundary_integral, pressure scalar1 = pressure_int
   print pressure_int, text = 'integral of pressure over curve c2'
 # Write the results to a file
   output
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c4), degfd1, quadratic   # The u-component of the velocity at
                                   # instream is quadratic
                                   # The rest of the vector is 0

end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1
```

```
coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef6  = 1d-12       # 6:  Penalty parameter to prevent singular matrix
      coef7  = rho         # 7:  Density
      coef12 = eta         #12:  Value of eta (viscosity)
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end
#
#  Define information with respect to the boundary integral to be computed
#  See Users Manual, Section 3.2.14
#
boundary_integral, sequence_number = 1
   ichint = 1                 # Standard integration
   curves = c4                # integral over curve c4
end

# compute pressure
# See Users Manual, Section 3.2.11

derivatives, sequence_number = 1
   icheld=7          # icheld=7, pressure in nodes
                     # See Standard problems Section 7.1
end

end_of_sepran_input
```

You can see that in this case we have introduced a penalty function parameter. The reason is that the matrix is singular if we set the penalty function parameter equal to zero. This is caused by the fact that this element does not satisfy the BB condition.

Adding a very small amount of penalty function, which means the diagonal of the matrix corresponding to the pressure rows is updated by a small number, is sufficient to get rid of this singularity.

Integrated method with elimination A special possibility is to use shape number 7 in combination with the elimination of the centroid velocity and the gradient of the pressure in the element centers. In this case type number 901 must be used. Furthermore there is no difference with type number 902.
The mesh input file is given by:

```
#  channel31.msh
#
#  mesh file for 2d channel problem
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepmesh channel31.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the channel
      length = 4             # length of the channel
   integers
      n = 4                  # number of elements in length direction
      m = 4                  # number of elements in width direction
      shape_cur = 2          # Type of elements along curves
                             # quadratic elements
      shape_sur = 7          # Type of elements in surface
                             # Extended quadratic triangles
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)               # Left under point
      p2=(length,0)          # Right under point
      p3=(length,width)      # Right upper point
      p4=(0,width)           # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line shape_cur (p1,p2,nelm=n)     # lower wall
      c2=line shape_cur (p2,p3,nelm=m)     # outflow boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper wall
      c4=line shape_cur (p4,p1,nelm=m)     # inflow boundary
```

```
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle shape_sur (c1,c2,c3,c4)

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2
   renumber, start = c2, Cuthill_McKee, always
                                  # Force a renumbering
                                  # See Users Manual Section 2.2

end
```

the problem input file is:

```
# channel31.prb
#
#  problem file for 2d channel problem
#  integrated method, centroid velocity and pressure gradient eliminated
#  problem is stationary and non-linear
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepcomp channel31.prb
#
#  Reads the file meshoutput
#  Creates the files sepcomp.inf and sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
   vector_names
      velocity
      pressure
   variables
      pressure_int
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=901               # Type number for Navier-Stokes, without swirl
                               # integrated approach
                               # See Standard problems Section 7.1
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
```

```
      curves(c1)               # Fixed under wall
      curves(c3)               # Fixed side walls and instream boundary
      curves(c4)               # inflow
      degfd2=curves(c2)        # Outstream boundary (v-component given)
                               # All not prescribed boundary conditions
                               # satisfy corresponding stress is zero
   renumber levels (1,2),(3)   # renumber the unknowns such that for each
                               # level first we have all velocities and then
                               # all pressures, thus avoiding zero pivots
end
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because the integral of the pressure over the boundary
#  is required
#
structure                    # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions, velocity
   solve_nonlinear_system, velocity
 # Compute the pressure
   derivatives, pressure
 # Compute the integral of the pressure over curve c2 (outflow boundary)
   boundary_integral, pressure, scalar1 = pressure_int
   print pressure_int, text = 'integral of pressure over curve c2'
 # Write the results to a file
   output
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c4), degfd1, quadratic   # The u-component of the velocity at
                                   # instream is quadratic
                                   # The rest of the vector is 0

end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef7  = rho         # 7:  Density
      coef12 = eta         #12:  Value of eta (viscosity)
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7
```

```
change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end
#
#  Define information with respect to the boundary integral to be computed
#  See Users Manual, Section 3.2.14
#
boundary_integral, sequence_number = 1
   ichint = 1                 # Standard integration
   curves = c4                # integral over curve c4
end

# compute pressure
# See Users Manual, Section 3.2.11

derivatives, sequence_number = 1
   icheld=7          # icheld=7, pressure in nodes
                     # See Standard problems Section 7.1
end

end_of_sepran_input
```

Taylor Hood elements Taylor Hood elements are characterized by the fact that the pressure is not longer discontinuous but that a continuous approximation with unknowns in the vertices is applied.
In this case type number 903 must be used.
At this moment 4 different shapes of elements are available.

shape = 3 This is the so-called mini element. Both the velocity and the pressure are approximated linearly. However, the velocity field consists of a linear part plus a bubble function that is eliminated later on.
Since the pressure is available in the vertices, one could think of prescribing the pressure at the outflow. However, mathematically speaking one should not prescribe the pressure explicitly but use the normal stress instead. In fact prescribing the pressure does not give essentially different results.
The mesh input file is given by

```
#  channel41.msh
#
```

```
#  mesh file for 2d channel problem
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepmesh channel41.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the channel
      length = 4             # length of the channel
   integers
      n = 8                  # number of elements in length direction
      m = 8                  # number of elements in width direction
      shape_cur = 1          # Type of elements along curves
                             # linear elements
      shape_sur = 3          # Type of elements in surface
                             # Linear triangles (mini element)
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)               # Left under point
      p2=(length,0)          # Right under point
      p3=(length,width)      # Right upper point
      p4=(0,width)           # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line shape_cur (p1,p2,nelm=n)     # lower wall
      c2=line shape_cur (p2,p3,nelm=m)     # outflow boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper wall
      c4=line shape_cur (p4,p1,nelm=m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle shape_sur (c1,c2,c3,c4)

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2
   renumber, start = c2, Cuthill_McKee, always
                                  # Force a renumbering
                                  # See Users Manual Section 2.2
```

```
end
```

And the corresponding problem input file:

```
# channel41.prb
#
#  problem file for 2d channel problem
#  integrated method
#  problem is stationary and non-linear
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepcomp channel41.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
   vector_names
      velocity_pressure
   variables
      pressure_int
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=903                # Type number for Navier-Stokes, without swirl
                                # integrated approach, Taylor Hood approach
                                # See Standard problems Section 7.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
                                # Only velocities are prescribed, not the
                                # pressures
      degfd1,degfd2=curves(c1) # Fixed under wall
      degfd1,degfd2=curves(c3) # Fixed side walls and instream boundary
      degfd1,degfd2=curves(c4) # inflow
      degfd2       =curves(c2) # Outstream boundary (v-component given)
                                # All not prescribed boundary conditions
                                # satisfy corresponding stress is zero
end
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because the integral of the pressure over the boundary
#  is required
```

```
#
structure                    # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions, velocity_pressure
   solve_nonlinear_system, velocity_pressure
 # Compute the integral of the pressure over curve c2 (outflow boundary)
 # Now the pressure is part of the solution vector
   boundary_integral, velocity_pressure scalar1 = pressure_int
   print pressure_int, text = 'integral of pressure over curve c2'
 # Write the results to a file
   output
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c4), degfd1, quadratic   # The u-component of the velocity at
                                   # instream is quadratic
                                   # The rest of the vector is 0

end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef7  = rho         # 7:  Density
      coef12 = eta         #12:  Value of eta (viscosity)
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1
```

```
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end
#
#  Define information with respect to the boundary integral to be computed
#  See Users Manual, Section 3.2.14
#
boundary_integral
   ichint = 1                # Standard integration
   curves = c4               # integral over curve c4
   degree_of_freedom = 3     # The pressure is third degree of freedom
end

end_of_sepran_input
```

Note that in this case it is necessary to prescribe explicitly the degrees of freedom 1 and 2 at boundaries where the velocity is given, since the third degree of freedom corresponds to the pressure. The pressure is not prescribed at the boundary. Since the pressure is already available in the vertices, there is no need to write the pressure separately to the output file. However, using the output option as in the case of the Crouzeix Raviart elements is also allowed.
The corresponding postprocessing file is

```
#  channel41.pst
#  Input file for postprocessing for channel problem
#  See Manual Standard Elements Section 7.1.8
#
#
#  To run this file use:
#     seppost channel41.pst > channel41.out
#
#  Reads the files meshoutput and sepcomp.out
#
#
postprocessing                    # See Users Manual Section 5.2
#
#  compute the stream function
#  See Users Manual Section 5.2
#  store in stream_function

   compute stream_function = stream function velocity_pressure

#  Plot the results
#  See Users Manual Section 5.4

   plot vector velocity_pressure              # Vector plot of velocity
   plot contour velocity_pressure, degfd=3    # Contour plot of pressure
   plot coloured contour velocity_pressure, degfd=3
   plot contour stream_function             # Contour plot of stream function
   plot coloured contour stream_function

end
```

The quality of the solution in this case is less than that of the other elements. The

velocity field looks al-right but the pressure contours (Figure 7.1.8.8) are definitely less accurate. The only reason to use this element is that it has only a limited number of unknowns and that it can be used easily in combination with iterative linear solvers.

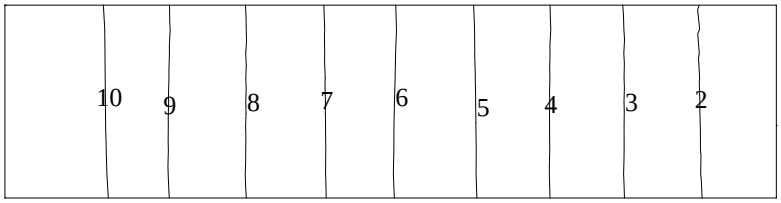


Figure 7.1.8.8: Isobars (mini element shape=3)

shape = 4 The quadratic element is of course exact.
The mesh input file is given by

```
#  channel42.msh
#
#  mesh file for 2d channel problem
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepmesh channel42.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the channel
      length = 4             # length of the channel
   integers
      n = 4                  # number of elements in length direction
      m = 4                  # number of elements in width direction
      shape_cur = 2          # Type of elements along curves
                             # quadratic elements
      shape_sur = 4          # Type of elements in surface
                             # quadratic triangles
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)              # Left under point
```

```
      p2=(length,0)         # Right under point
      p3=(length,width)    # Right upper point
      p4=(0,width)          # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line shape_cur (p1,p2,nelm=n)     # lower wall
      c2=line shape_cur (p2,p3,nelm=m)     # outflow boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper wall
      c4=line shape_cur (p4,p1,nelm=m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle shape_sur (c1,c2,c3,c4)

   plot                            # make a plot of the mesh
                                   # See Users Manual Section 2.2
   renumber, start = c2, Cuthill_McKee, always
                                   # Force a renumbering
                                   # See Users Manual Section 2.2

end
```

The input files for SEPCOMP and SEPPOST are identical to the ones for the mini element.

shape = 6 Also in this case we have an exact solution.
The mesh input file is given by:

```
#  channel43.msh
#
#  mesh file for 2d channel problem
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepmesh channel43.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the channel
      length = 4             # length of the channel
   integers
      n = 4                  # number of elements in length direction
      m = 4                  # number of elements in width direction
      shape_cur = 2          # Type of elements along curves
                             # quadratic elements
      shape_sur = 6          # Type of elements in surface
                             # Bi-quadratic quadrilaterals
end
#
```

```
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)              # Left under point
      p2=(length,0)         # Right under point
      p3=(length,width)    # Right upper point
      p4=(0,width)          # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line shape_cur (p1,p2,nelm=n)     # lower wall
      c2=line shape_cur (p2,p3,nelm=m)     # outflow boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper wall
      c4=line shape_cur (p4,p1,nelm=m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle shape_sur (c1,c2,c3,c4)

   plot                             # make a plot of the mesh
                                    # See Users Manual Section 2.2
   renumber, start = c2, Cuthill_McKee, always
                                    # Force a renumbering
                                    # See Users Manual Section 2.2

end
```

shape = 10 This element is equivalent to the mini element. The only difference is that the mid point has not been eliminated.
The mesh input file is:

```
#  channel44.msh
#
#  mesh file for 2d channel problem
#  See Manual Standard Elements Section 7.1.8
#
#  To run this file use:
#     sepmesh channel44.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the channel
      length = 4             # length of the channel
   integers
      n = 8                  # number of elements in length direction
```

```
      m = 8                  # number of elements in width direction
      shape_cur = 1          # Type of elements along curves
                             # linear elements
      shape_sur = 10         # Type of elements in surface
                             # Extended linear triangles
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)               # Left under point
      p2=(length,0)          # Right under point
      p3=(length,width)     # Right upper point
      p4=(0,width)           # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line shape_cur (p1,p2,nelm=n)     # lower wall
      c2=line shape_cur (p2,p3,nelm=m)     # outflow boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper wall
      c4=line shape_cur (p4,p1,nelm=m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle shape_sur (c1,c2,c3,c4)

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2
   renumber, start = c2, Cuthill_McKee, always
                                  # Force a renumbering
                                  # See Users Manual Section 2.2

end
```

7.1.9 Example of a periodic channel flow

In this section we consider a simple channel flow (Cartesian co-ordinates) for low Reynolds numbers. This is the same problem as treated in Section 7.1.8. However, in this case we prescribe the mass flux at the inflow boundary C4, see Figure 7.1.8.1 and we assume that velocity is periodical at sides C2 and C4.
As a consequence the pressure at inflow and outflow will also be periodical, however, with an unknown pressure difference. This difference is implicitly defined by the mass flux.
To solve this problem both the penalty function approach (elements of type 912) and the approach with global unknowns (elements of type 913) is considered.
Just as in Section 7.1.8 there are a number of examples available.
In order to get these examples into your local directory use the command

```
sepgetex chanperx
```

where x is a 1 digit number. The following numbers are available:

number	shape	type	description
1	4	900	extended quadratic triangle, penalty method, penalty approach
2	4	900	extended quadratic triangle, penalty method, global unknowns
3	7	902	extended quadratic triangle, integrated method, global unknowns
4	7	902	See 3, iterative linear solver
5	3	903	linear triangle, Taylor Hood, global unknowns
6	3	902	See 5, iterative linear solver
7	6	901	biquadratic quadrilateral, Taylor Hood, global unknowns
8	6	903	See 7, iterative linear solver

penalty function approach

In order to get this example into your local directory use the command

```
sepgetex chanper1
```

The mesh definition is nearly the same as in 7.1.8, except for two items. First of all we need to define a line element along C4, that is used to define the mass flux. Next we need connection elements to define the periodical boundary conditions.
The input file for SEPMESH (chanper.msh) has the following form:

```
#  chanper1.msh
#
#  mesh file for 2d channel problem
#  periodical boundary conditions
#  penalty function approach
#  Mass flux given, treated with large line element and penalty approach
#  Crouzeix-Raviart type elements
#  See Manual Standard Elements Section 7.1.9
#
#  To run this file use:
#     sepmesh chanper1.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the channel
      length = 4             # length of the channel
```

```
   integers
      n = 4                  # number of elements in length direction
      m = 4                  # number of elements in width direction
      shape_cur = 2          # Type of elements along curves
                             # quadratic elements
      shape_sur = 4          # Type of elements in surface
                             # quadratic triangles
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)               # Left under point
      p2=(length,0)          # Right under point
      p3=(length,width)      # Right upper point
      p4=(0,width)           # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line shape_cur (p1,p2,nelm=n)     # lower wall
      c2=line shape_cur (p2,p3,nelm=m)     # outflow boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper wall
      c4=line shape_cur (p4,p1,nelm=m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1=rectangle shape_sur (c1,c2,c3,c4)
   meshline
      lelm1 = (shape=-1,c4)              # One large line element for the
                                         # mass flux
   meshsurf
      selm2=s1                           # Internal elements
   meshconnect
      celm3 = curves300(c2,-c4)          # Connection elements for the
                                         # periodical boundary conditions

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2

end
```

To run program SEPCOMP we need an input file. Instead of the usual one element group as in Section 7.1.8, we need 3 groups.

element group 1 corresponds to the line element and has type number 912. This defines the mass flux.

element group 2 corresponds to the internal elements and has type number 900. This defines the Navier-Stokes equations.

element group 3 corresponds to the connection elements and has type number -1. This defines the periodical boundary conditions.

Furthermore in the computation of the pressure it is necessary to skip over the periodical boundary elements, since otherwise the pressure is also made periodical. This means that we can not define the pressure in the input block OUTPUT but need a separate block DERIVATIVES.
As a consequence a block STRUCTURE is necessary, since otherwise the derivatives block is never used.

The input file for SEPCOMP looks like:

```
# chanper1.prb
#
#  problem file for 2d channel problem
#  periodical boundary conditions
#  penalty function approach
#  Mass flux given, treated with large line element and penalty approach
#  Crouzeix-Raviart type elements
#  problem is stationary and non-linear
#  See Manual Standard Elements Section 7.1.9
#
#  To run this file use:
#     sepcomp chanper1.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      massflux  = 0.66666667         # mass flux
      penalflux = 1d6                # penalty parameter for mass flux
      eps       = 1d-6               # penalty parameter for Navier-Stokes
      rho       = 1                  # density
      eta       = 0.01               # viscosity
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=912               # Type number for given mass flux
      elgrp2=900               # Type number for Navier-Stokes, without swirl
      elgrp3=-1                # periodic boundary conditions
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
```

```
      curves(c1)              # Fixed under wall
      curves(c3)              # Fixed upper wall
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary since the computation of the pressure requires some
#  extra care
#
structure                   # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions, velocity
   solve_nonlinear_system, velocity
 # Compute the pressure
   derivatives, pressure
 # Write the results to a file
   output
end

#  Define the coefficients for the problems
#  See Users Manual Section 3.2.6

coefficients
   elgrp1 ( nparm=10 )     # The coefficients for the mass flux bc
                           # are defined by 10 parameters
      icoef3 = 2           # 3:  type of integration (2=quadratic)
      icoef5 = 1           # 5:  Degree of freedom (1=u)
      coef6  = massflux    # 6:  Mass flux
      coef7  = penalflux   # 7:  Penalty parameter
   elgrp2 ( nparm=20 )     # The coefficients for Navier-Stokes are defined
                           # by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef6  = eps         # 6:  Penalty function parameter eps
      coef7  = rho         # 7:  Density
      coef12 = eta         #12:  Value of eta (viscosity)
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp2
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp2
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver

nonlinear_equations     # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1
```

```
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

#  The pressure is computed as a derived quantity of the Navier-Stokes
#  equation
#  See Users Manual Section 3.2.11

derivatives
   icheld = 7
   skip_element_groups = (3)       # The pressure is not periodic and hence
                                   # this group must be skipped
end
end_of_sepran_input
```

The postprocessing file is in this case exactly the same as for the standard channel flow problem:

```
#  chanper1.pst
#  Input file for postprocessing for channel problem
#  periodical boundary conditions
#  penalty function approach
#  Mass flux given, treated with large line element and penalty approach
#  Crouzeix-Raviart type elements
#  See Manual Standard Elements Section 7.1.9
#
#
#  To run this file use:
#     seppost chanper1.pst > chanper1.out
#
#  Reads the files meshoutput and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2
#
#  compute the stream function
#  See Users Manual Section 5.2
#
   compute stream function velocity

#  Plot the results
#  See Users Manual Section 5.4

   plot vector velocity             # Vector plot of velocity
   plot contour pressure text='isobars' # Contour plot of pressure
   plot coloured contour pressure
   plot contour stream_function          # Contour plot of stream function
   plot coloured contour stream_function
end
```

The results produced are identical to the ones shown in Section 7.1.8 and will not be repeated.

global unknowns approach

In order to get this example into your local directory use the command

```
   sepgetex chanper2
```

In this case there is no need to introduce a large line element. Only the periodical boundary conditions are needed and hence the connection elements. The mesh input file is for example

```
#  chanper2.msh
#
#  mesh file for 2d channel problem
#  periodical boundary conditions
#  Mass flux given, treated with global unknowns
#  Crouzeix-Raviart type elements
#  See Manual Standard Elements Section 7.1.9
#
#  To run this file use:
#     sepmesh chanper2.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1              # width of the channel
      length = 4             # length of the channel
   integers
      n = 4                  # number of elements in length direction
      m = 4                  # number of elements in width direction
      shape_cur = 2          # Type of elements along curves
                             # quadratic elements
      shape_sur = 4          # Type of elements in surface
                             # quadratic triangles
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)              # Left under point
      p2=(length,0)         # Right under point
      p3=(length,width)     # Right upper point
      p4=(0,width)          # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line shape_cur (p1,p2,nelm=n)     # lower wall
      c2=line shape_cur (p2,p3,nelm=m)     # outflow boundary
      c3=line shape_cur (p3,p4,nelm=n)     # upper wall
      c4=line shape_cur (p4,p1,nelm=m)     # inflow boundary
#
#  surfaces
#
```

```
   surfaces         # See Users Manual Section 2.4
      s1=rectangle shape_sur (c1,c2,c3,c4)
   meshsurf
      selm1=s1                          # Internal elements
   meshconnect
      celm2 = curves300(c2,-c4)         # Connection elements for the
                                        # periodical boundary conditions

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2

end
```

In the problem definition we have to introduce one global unknown, representing the pressure jump. This global unknown corresponds to the mass flux and is defined over the inflow boundary c4.
So now we have two element groups and one global element group.

element group 1 corresponds to the internal elements and has type number 900. This defines the Navier-Stokes equations.

element group 2 corresponds to the connection elements and has type number -1. This defines the periodical boundary conditions.

global element group 1 corresponds to the curve c4 and has type number 913. This defines the mass flux.

The rest of the input is more or less the same as for the penalty approach.
The input file for SEPCOMP is:

```
# chanper2.prb
#
#  problem file for 2d channel problem
#  periodical boundary conditions
#  penalty function approach
#  Mass flux given, treated with global unknowns
#  Crouzeix-Raviart type elements
#  problem is stationary and non-linear
#  See Manual Standard Elements Section 7.1.9
#
#  To run this file use:
#     sepcomp chanper2.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      massflux  = 0.66666667         # mass flux
      eps       = 1d-6               # penalty parameter for Navier-Stokes
      rho       = 1                  # density
      eta       = 0.01               # viscosity
   vector_names
```

```
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=900               # Type number for Navier-Stokes, without swirl
      elgrp2=-1                # periodic boundary conditions
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      curves(c1)               # Fixed under wall
      curves(c3)               # Fixed upper wall
   global_unknowns             # define element group for global unknown
      glgrp1=913               # Type number for given mass flux
   global_elements
      gelm1 = curves(c4)       # mass flux is defined along inflow boundary
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary since the computation of the pressure requires some
#  extra care
#
structure                    # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions, velocity
   solve_nonlinear_system, velocity
 # Compute the pressure
   derivatives, pressure
 # Write the results to a file
   output
end

#  Define the coefficients for the problems
#  See Users Manual Section 3.2.6

coefficients
   elgrp1 ( nparm=20 )     # The coefficients for Navier-Stokes are defined
                           # by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef6  = eps         # 6:  Penalty function parameter eps
      coef7  = rho         # 7:  Density
      coef12 = eta         #12:  Value of eta (viscosity)
   glgrp1 ( nparm=10 )     # The coefficients for the mass flux bc
                           # are defined by 10 parameters
      icoef5 = 1           # 5:  Degree of freedom (1=u)
      coef6  = massflux    # 6:  Mass flux
end
```

```
#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver

nonlinear_equations, sequence_number = 1     # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

#  The pressure is computed as a derived quantity of the Navier-Stokes
#  equation
#  See Users Manual Section 3.2.11

derivatives
   icheld = 7
   skip_element_groups = (2)       # The pressure is not periodic and hence
                                   # this group must be skipped
end
end_of_sepran_input
```

The post processing file is exactly the same as for the penalty approach and is not repeated here.

7.1.10 Flow between staggered pipes with anti-symmetric boundary conditions

In this section we consider the flow between a number of pipes in a staggered arrangement. Due to the staggering of the pipes it is sufficient to consider the dashed region in Figure 7.1.10.1. At

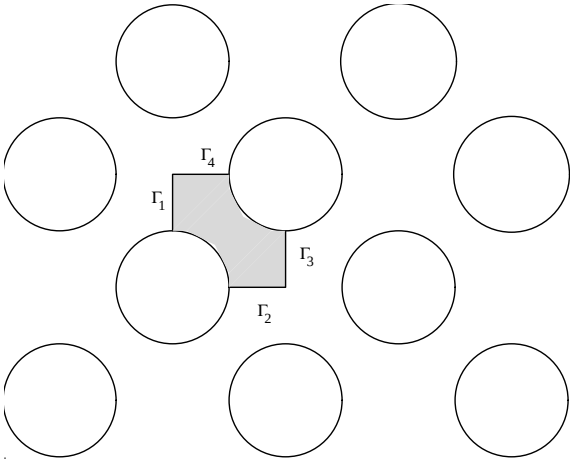


Figure 7.1.10.1: Computational region in array of staggered pipes with anti-periodic boundary conditions

the boundaries Γ_2 and Γ_4 symmetry boundary conditions are used, i.e. $\mathbf{u} \cdot \mathbf{n} = 0, \sigma^{nt} = 0$ and along the boundaries Γ_1 and Γ_3 we need anti-symmetrical boundary conditions. This means that the velocity is anti-symmetric periodical and that the pressure has a pressure difference. In fact the same method as in Section 7.1.9 is used, with the exception that points at sides Γ_1 and Γ_3 are connected in crossed way. Hence points at the top of Γ_1 are connected with points at the lower part of Γ_3 and vice versa.

The example we use is described in Segal et al (1994).
The radius of the pipes is 10.85 mm, the distance between the centroids of neighboring pipes is 45 mm both in horizontal as in vertical directions. The mean velocity V_0 (from left to right) at the inlet is 1.06 m/s, which implies that the flow rate Q is given by $Q = 0.01235\ m^3/s$. The Reynolds number Re_D is related to the diameter D of the pipes. The flow has been computed for $Re_D = \frac{\rho V_0 D}{\mu} \approx 362$.

In order to get this example into your local directory use the command

```
sepgetex tube
```

The definition of the curves is given in Figure 7.1.10.2.
The input file for SEPMESH (tube.msh) has the following form:

```
*
*  tube.msh
*
* mesh input for the staggered pipes example
*
mesh2d
   coarse ( unit=0.001)
   points
      p1=(-0.01165,0)
      p2=(0,0)
```

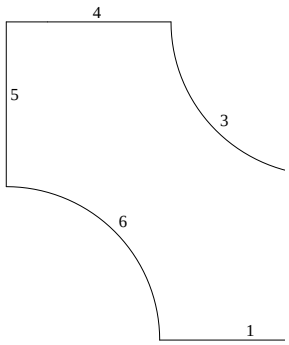


Figure 7.1.10.2: Definition of curves for tube problem

```
      p3=(0,0.01165)
      p4=(-0.01085,0.0225)
      p5=(-0.0225,0.0225)
      p6=(-0.0225,0.01085)
      p7=(-0.0225,0)
      p8=(0,0.0225)
   curves
      c1=cline2(p1,p2)
      c2=cline2(p2,p3)
      c3=carc2(p3,p4,-p8)
      c4=cline2(p4,p5)
      c5=cline2(p5,p6)
      c6=carc2(p6,p1,-p7)
   surfaces
      s1=general4(c1,c2,c3,c4,c5,c6)
   meshline
      lelm1 = (shape=-1,c5)              # Line element for mass flux
   meshsurf
      selm2=s1                           # Internal element
   meshconnect
      celm3 = curves0(c2,c5)             # Connection elements for
                                         # anti-symmetric periodical
                                         # boundary conditions
   plot

end
```

The mesh created including the connection elements is shown in Figure 7.1.10.3.
The input file for SEPCOMP is nearly the same as the one described in Section 7.1.10

```
*  tube.prb
*
*  input for computing program Navier-Stokes in staggered pipes with
*  anti-symmetrical periodical boundary conditions
*
*  Penalty function method
*
*
```

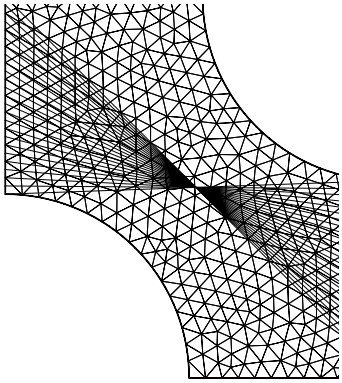


Figure 7.1.10.3: Mesh for tube problem

```
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   vector_names
      velocity
      pressure
end

problem

* Define type of elements

   types

      elgrp1=912               # Type number for given mass flux
      elgrp2=900               # Type number for Navier-Stokes, without swirl
      elgrp3=-1                # periodic boundary conditions

*   Define where essential boundary conditions are present

   essbouncond
      curves (c3)                 # Fixed upper tube
      curves (c6)                 # Fixed lower tube
      degfd2, curves (c4)         # v=0 at c4, (symmetry)
      degfd2, curves (c1)         # v=0 at c1, (symmetry)
end

* Define structure of the program
* This is necessary since the compuation of the pressure requires some
* extra care

structure
*  Compute the velocity
   prescribe_bounday_conditions, velocity
   solve_nonlinear_system, velocity
*  Compute the pressure
```

```
   derivatives, pressure
   output
end

* Input for subroutine FILCOF at the first iteration (iteration 0)
* At this moment the input for FILCOF is required for each iteration
* with a change in the input.
* In a forthcoming version it will not longer be necessary to repeat this
* input completely

coefficients
   elgrp1 ( nparm=10 )     # The coefficients for the mass flux bc
                           #are defined by 10 parameters
      icoef3 = 2           # 3:  type of integration (2=quadratic)
      icoef5 = 1           # 5:  Degree of freedom (1=u)
      coef6  = 0.01235     # 6:  Mass flux
      coef7  = 1d6         # 7:  Penalty parameter
   elgrp2 ( nparm=20 )     # The coefficients for Navier-Stokes are defined
                           # by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef6  = 1d-6        # 6:  Penalty function parameter eps
      coef7  = 1           # 7:  Density
      coef12 = 0.000635    #12:  Value of eta (viscosity)
end

*  Define the coefficients for the next iterations

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp2
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp2
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end


* Define the parameters for the non-linear solver

nonlinear_equations
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

* Define the computation of the pressure

derivatives
   icheld = 7
   skip_element_groups = (3)       # The pressure is not periodic and hence
```

```
                                    # this group must be skipped
end
end_of_sepran_input
```

The postprocessing file is in this case exactly the same as for the standard channel flow problem:

```
*
*  tube.pst
*
post processing

*  Compute stream funnction, store in stream_function, and name this vector

   compute stream_function = stream function velocity

*  PLot the results

   plot vector velocity              # Vector plot of velocity
   plot contour pressure              # Contour plot of pressure
   plot coloured contour pressure
   plot contour stream_function             # Contour plot of stream function
   plot coloured contour stream_function
end
```

Figure 7.1.10.4 shows the computed isobars, Figure 7.1.10.5 a colored levels plot of the pressure, Figure 7.1.10.6 the stream lines and, Figure 7.1.10.7 a colored levels plot of the stream function.

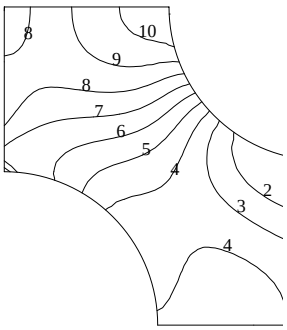


Figure 7.1.10.4: Isobars for tube problem

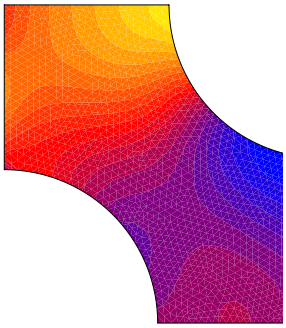


Figure 7.1.10.5: Colored pressure levels for tube problem

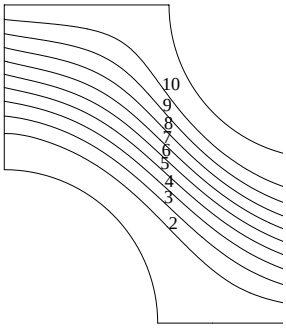


Figure 7.1.10.6: Stream lines for tube problem

7.1.11 Example of flow in a tube

In this section we consider the flow in a tube for low Reynolds numbers. This problem has been provided by Peter Dierickx of Gent University.
This problem is very similar to that in Sections 7.1.8 and 7.1.9, however instead of a channel we consider a tube.
First we consider the problem as a 2d axi-symmetric flow, later on we solve it as a three-dimensional problem. There are several ways to solve the problem, all of which must give more or less the same result.
The region in which the problem must be solved is sketched in Figure 7.1.11.1. The water flows from the lower face (S6) to the upper face (S7). Since the problem is axi-symmetric it can be solved as an axi-symmetric flow and then it is sufficient to consider the region sketched in Figure 7.1.11.2 (r,z-plane).
The problem can be solved analytically resulting in a quadratic velocity profile. In this section we try to solve the problem in 5 different ways:

- As an axi-symmetric problem by prescribing the inflow velocity.
- As an axi-symmetric problem by prescribing the mass flux in combination with the penalty function approach.

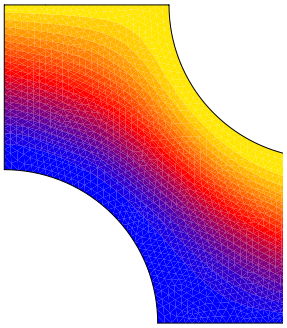


Figure 7.1.10.7: Colored stream function levels for tube problem

- As an axi-symmetric problem by prescribing the mass flux in combination with a global unknown.
- As a 3d problem by prescribing the mass flux in combination with a global unknown.
- Again as a 3d problem by prescribing the mass flux in combination with a global unknown. In this case however, we use local transformations.

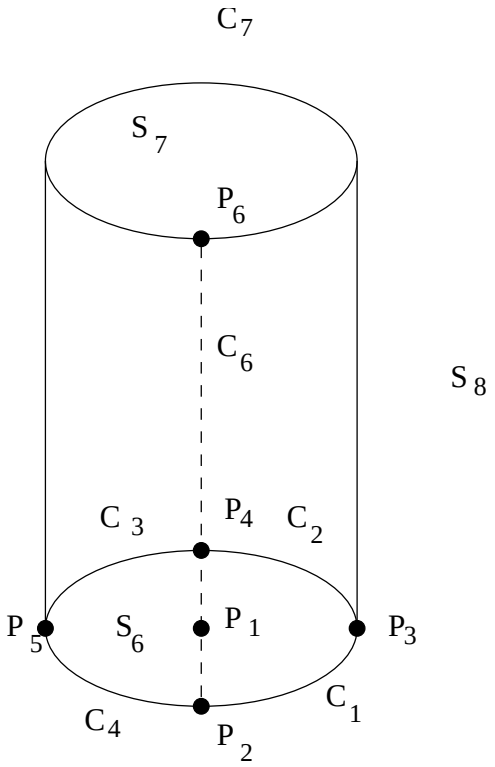


Figure 7.1.11.1: Definition of tube with generating surfaces

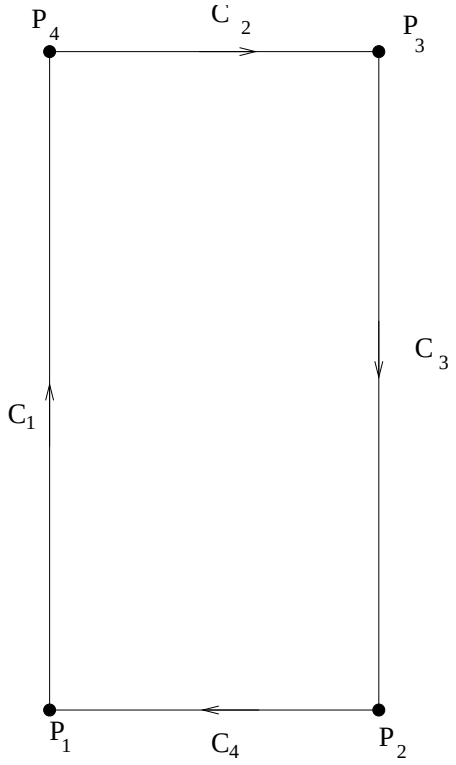


Figure 7.1.11.2: Definition of cross-section of tube with generating curves

7.1.11.1 Axi-symmetric with inflow velocity field

In order to get this example into your local directory use the command

```
sepgetex tubeax1
```

The region to be defined is sketched in Figure 7.1.11.2. The height of the tube is 40, the radius is 20.
The input file for SEPMESH (tubeax1.msh) has the following form:

```
#  tubeax1.msh
#  stationary laminar Newtonian flow
#  mesh file for axi-symmetric flow of water in a tube
#
#  See Manual Examples Section 7.1.11
#
#  To run this file use:
#     sepmesh tubeax1.msh
#
#  Creates the file meshoutput
#
#  Define some rectangle constants
#
constants            # See Users Manual Section 1.4
   real
      r = 20         # radius = 20
      l = 40         # length = 40
   integers
      nel_len = 4       # number of elements in length direction
      nel_rad = 3       # number of elements in radial direction
end
#
#  Define the mesh
#
mesh2d               # See Users Manual Section 2.2
#
#  user points
#
   points            # See Users Manual Section 2.2
      p1 = (0,0)     # Left under point
      p2 = ( r,0)    # Right under point
      p3 = ( r, l)   # Right upper point
      p4 = (0, l)    # Left upper point
#
#  curves
#
   curves            # See Users Manual Section 2.3
                     # Quadratic elements are used
      c1 = line2(p1,p4,nelm= nel_len, ratio=1, factor=1)  # symmetry axis
      c2 = line2(p4,p3,nelm= nel_rad, ratio=1, factor=1)  # outflow
      c3 = line2(p3,p2,nelm= nel_len, ratio=1, factor=1)  # fixed wall
      c4 = line2(p2,p1,nelm= nel_rad, ratio=1, factor=1)  # inflow
#
#  surfaces
#
   surfaces          # See Users Manual Section 2.4
```

```
                    # Quadratic quadrilaterals are used
      s1 = rectangle6(c1,c2,c3,c4)

   plot                            # make a plot of the mesh
                                   # See Users Manual Section 2.2

end
```

The v-component of the inflow velocity is given by the function: $v(r) = 20(1 - \frac{r^2}{20^2})$. In order to introduce this velocity it is necessary to define a function subroutine FUNCBC. For that reason we need a main program tubeax1.f:

```
      program tubeax1

!     --- Main program for axi-symmetric flow of water in a tube
!         Periodical boundary conditions
!         To link this program use:
!
!         seplink tubeax1

      implicit none
      call sepcom (0)
      end

!     --- Define the velocity at inflow (quadratic profile)

      function funcbc ( ichois, x, y, z )
      implicit none
      integer ichois
      double precision x, y, z, funcbc

      if ( ichois==1 ) then

         funcbc = 20d0 * ( 1d0 -x**2 / 20d0**2 )

      end if

      end
```

To run program tubeax1 we need an input file. In this special case it has been decided to prescribe the normal stress (pressure) at the outflow boundary. Therefore boundary elements of type 910 are used. The rest of the input is more or less standard.

The input file for tubeax1 looks like:

```
# tubeax1.prb
#
#  problem file for the axi-symmetric flow of water in a tube
#  stationary laminar Newtonian flow
#  penalty function approach
#  See Manual Examples Section 7.1.11
#
#  To run this file use:
#     sepcomp tubeax1.prb
#
#  Reads the file meshoutput
```

```
#  Creates the files sepcomp.inf and sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1d-6              # penalty parameter for Navier-Stokes
      rho       = 998.2d-6          # density [g/mm3]
      eta       = 1.002d-3          # viscosity [g/mm/s] [Pa.s]
      p0        = 10                # pressure at outflow
   vector_names
      velocity
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
      elgrp1 = 900              # type number for Navier-Stokes without swirl
   natbouncond                  # Define type numbers for boundary elements
      bngrp 1 = 910             # type number for given stress for Nav-Stokes
   bounelements                 # Define where boundary elements are given
      belm1 = curves (shape = 2, c2)    # stress given at outflow boundary
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
      curves (c3)               # fixed side wall
      degfd1, curves (c1)       # symmetry axis, u=0
      curves (c4)               # inflow, velocity given
      degfd1, curves (c2)       # fully developed flow (u=0)
end

#  Fill the non-zero values of the essential boundary conditions
#  See Users Manual Section 3.2.5

essential boundary conditions
   curves(c4), degfd2, func = 1   # v is given by a function
end

#  Define the coefficients for the problems
#  See Users Manual Section 3.2.6
#  See also standard problems Section 7.1

coefficients
   elgrp1 (nparm = 20)      # coefficients for Navier-Stokes
      icoef4 = 1            # axi-symmetric co-ordinates
      icoef5 = 0            # stokes flow, neglecting convective terms v.v
      coef6 =  eps          # penalty parameter
      coef7 =  rho          # density
      coef12 =  eta         # viscosity
   bngrp1 (nparm = 15)      # coefficients for the prescribed stress
      icoef1 = 1            # prescribed stresses normal and tangential to boundary
      icoef4 = 1            # axi-symmetric co-ordinates
```

```
      coef6 =  p0           # normal stress = pressure
      coef7 = 0             # tangential stress = 0 fully developed flow
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number=1 # input for iteration 2
   elgrp1
      icoef5 = 1           # Picard's linearization
end

change coefficients, sequence_number=2 # input for iteration 3
   elgrp1
      icoef5 = 2           # Newton's linearization
end

# input for non-linear solver

nonlinear_equations          # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-3,print_level=2, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

end_of_sepran_input
```

The postprocessing file is in this case exactly the same as for the standard channel flow problem:

```
#  tubeax1.pst
#  Input file for postprocessing for the axi-symmetric flow of water in a tube
#  See Manual Examples Section 7.1.11
#
#
#  To run this file use:
#     seppost tubeax1.pst > tubeax1.out
#
#  Reads the files meshoutput, sepcomp.inf and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2
#
#   compute the stream function
#  See Users Manual Section 5.2
#
   compute stream_function = stream function velocity

#  Plot the results
#  See Users Manual Section 5.4

   open plot
   plot vector velocity, factor = 0.10     # Vector plot of velocity
   plot curves
```

```
   plot points
   close plot
   open plot
   plot contour stream_function            # Contour plot of stream function
   plot curves
   plot points
   close plot
end
```

The velocity computed is quadratic and does not make sense to repeat the pictures.

7.1.11.2 Axi-symmetric flow with given mass flux by penalty approach

In order to get this example into your local directory use the command

```
sepgetex tubeax2
```

In this case we need to introduce a large line element because of the combination of mass flux and penalty function method. See Sections 7.1.1 and 7.1.9.
Furthermore it is necessary to prescribe periodical boundary conditions.
The mesh input file is for example

```
#  tubeax2.msh
#  stationary laminar Newtonian flow
#  mesh file for axi-symmetric flow of water in a tube
#  Periodical boundary conditions
#  Given mass flow with unknown constant
#
#  See Manual Examples Section 7.1.11
#
#  To run this file use:
#     sepmesh tubeax2.msh
#
#  Creates the file meshoutput
#
#  Define some rectangle constants
#
constants           # See Users Manual Section 1.4
   real
      r = 20        # radius = 20
      l = 40        # length = 40
   integers
      nel_len = 4      # number of elements in length direction
      nel_rad = 3      # number of elements in radial direction
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1 = (0,0)    # Left under point
      p2 = ( r,0)   # Right under point
      p3 = ( r, l)  # Right upper point
      p4 = (0, l)   # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1 = line2(p1,p4,nelm= nel_len, ratio=1, factor=1)  # symmetry axis
      c2 = line2(p4,p3,nelm= nel_rad, ratio=1, factor=1)  # outflow
      c3 = line2(p3,p2,nelm= nel_len, ratio=1, factor=1)  # fixed wall
      c4 = line2(p2,p1,nelm= nel_rad, ratio=1, factor=1)  # inflow
#
```

```
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
                    # Quadratic quadrilaterals are used
      s1 = rectangle6(c1,c2,c3,c4)
#
#   Define element groups
#
   meshsurf             # surface elements for Navier-Stokes
      selm1 = (s1)
   meshconnect          # connection elements for periodical boundary conditions
      celm2 = curves0(c4,-c2)

   plot                            # make a plot of the mesh
                                   # See Users Manual Section 2.2

end
```

The mass flux corresponding to the given velocity field in `tubeax1` is equal to $2\pi \int_{C4} \mathbf{u} \cdot \mathbf{n} r dr$ = 4000 π. The input file for sepcomp is (see also Section 7.1.9):

```
# tubeax2.prb
#
#  problem file for the axi-symmetric flow of water in a tube
#  stationary laminar Newtonian flow
#  penalty function approach
#  Periodical boundary conditions
#  Given mass flow with unknown constant
#  See Manual Examples Section 7.1.11
#
#  To run this file use:
#     sepcomp tubeax2.prb
#
#  Reads the file meshoutput
#  Creates the files sepcomp.inf and sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1d-6              # penalty parameter for Navier-Stokes
      rho       = 998.2d-6          # density [g/mm3]
      eta       = 1.002d-3          # viscosity [g/mm/s] [Pa.s]
      massflux  = 4000 * pi         # The mass flux is equal to 4000 pi
   vector_names
      velocity
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
```

```
      elgrp1 = 900             # type number for Navier-Stokes without swirl
      elgrp2 = -1              # type number for periodical boundary conditions
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
      curves (c3)              # fixed side wall
      degfd1, curves (c1)      # symmetry axis, u=0
   global_unknowns             # define element group for global unknown
      glgrp1=913               # Type number for given mass flux
   global_elements
      gelm1 = curves(shape=2, c4)   # mass flux is defined along inflow boundary
end

#  Define the structure of the large matrix

matrix                        # See Users Manual Section 3.2.4
               nosplit        # Non-symmetrical profile matrix
                              # So a direct method will be applied
                              # The matrix may not be splitted
end


#  Define the coefficients for the problems
#  See Users Manual Section 3.2.6
#  See also standard problems Section 7.1

coefficients
   elgrp1 (nparm = 20)        # coefficients for Navier-Stokes
      icoef4 = 1              # axi-symmetric co-ordinates
      icoef5 = 0              # stokes flow, neglecting convective terms v.v
      coef6 =  eps            # penalty parameter
      coef7 =  rho            # density
      coef12 =  eta           # viscosity
   glgrp1 ( nparm=10 )       # The coefficients for the mass flux bc
                             # are defined by 10 parameters
      icoef4 = 1             # axi-symmetric co-ordinates
      icoef5 = 2             # 5:  Degree of freedom (2=v)
      coef6  =  massflux     # 6:  Mass flux
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number=1 # input for iteration 2
   elgrp1
      icoef5 = 1             # Picard's linearization
end

change coefficients, sequence_number=2 # input for iteration 3
   elgrp1
      icoef5 = 2             # Newton's linearization
end

# input for non-linear solver

nonlinear_equations          # See Users Manual Section 3.2.9
```

```
   global_options, maxiter=10, accuracy=1d-3,print_level=2, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

end_of_sepran_input
```

The post processing file is exactly the same as for the first approach and is not repeated here.

7.1.11.3 Axi-symmetric flow with given mass flux by global unknown

In order to get this example into your local directory use the command

```
sepgetex tubeax3
```

In this case there is no need to introduce a large line element. See Sections 7.1.1 and 7.1.9.
Furthermore it is necessary to prescribe periodical boundary conditions.
The mesh input file is for example

```
#  tubeax3.msh
#  stationary laminar Newtonian flow
#  mesh file for axi-symmetric flow of water in a tube
#  Periodical boundary conditions
#  Given mass flow with unknown constant
#
#  See Manual Examples Section 7.1.11
#
#  To run this file use:
#     sepmesh tubeax2.msh
#
#  Creates the file meshoutput
#
#  Define some rectangle constants
#
constants           # See Users Manual Section 1.4
   real
      r = 20        # radius = 20
      l = 40        # length = 40
   integers
      nel_len = 4      # number of elements in length direction
      nel_rad = 3      # number of elements in radial direction
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1 = (0,0)    # Left under point
      p2 = ( r,0)   # Right under point
      p3 = ( r, l)  # Right upper point
      p4 = (0, l)   # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1 = line2(p1,p4,nelm= nel_len, ratio=1, factor=1)  # symmetry axis
      c2 = line2(p4,p3,nelm= nel_rad, ratio=1, factor=1)  # outflow
      c3 = line2(p3,p2,nelm= nel_len, ratio=1, factor=1)  # fixed wall
      c4 = line2(p2,p1,nelm= nel_rad, ratio=1, factor=1)  # inflow
#
#  surfaces
```

```
#
   surfaces         # See Users Manual Section 2.4
                    # Quadratic quadrilaterals are used
      s1 = rectangle6(c1,c2,c3,c4)
#
#   Define element groups
#
   meshline             # Large line element for mass flux
      lelm1 = (shape=-1, c4)
   meshsurf             # surface elements for Navier-Stokes
      selm2 = (s1)
   meshconnect          # connection elements for periodical boundary conditions
      celm3 = curves0(c4,-c2)

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2

end
```

The input file for sepcomp is given by

```
# tubeax3.prb
#
#  problem file for the axi-symmetric flow of water in a tube
#  stationary laminar Newtonian flow
#  penalty function approach
#  Periodical boundary conditions
#  Given mass flow with unknown constant
#  See Manual Examples Section 7.1.11
#
#  To run this file use:
#     sepcomp tubeax3.prb
#
#  Reads the file meshoutput
#  Creates the files sepcomp.inf and sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1d-6              # penalty parameter for Navier-Stokes
      rho       = 998.2d-6          # density [g/mm3]
      eta       = 1.002d-3          # viscosity [g/mm/s] [Pa.s]
      massflux  = 4000 * pi         # The mass flux is equal to 4000 pi
   vector_names
      velocity
end
#
#  Define the type of problem to be solved
#
problem                  # See Users Manual Section 3.2.2

   types                      # Define types of elements,
      elgrp1 = 912            # type number for mass flux
```

```
      elgrp2 = 900             # type number for Navier-Stokes without swirl
      elgrp3 = -1              # type number for periodical boundary conditions
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
      curves (c3)              # fixed side wall
      degfd1, curves (c1)      # symmetry axis, u=0
end

#  Define the structure of the large matrix

matrix                         # See Users Manual Section 3.2.4
               nosplit         # Non-symmetrical profile matrix
                               # So a direct method will be applied
                               # The matrix may not be splitted
end


#  Define the coefficients for the problems
#  See Users Manual Section 3.2.6
#  See also standard problems Section 7.1

coefficients
   elgrp2 (nparm = 20)         # coefficients for Navier-Stokes
      icoef4 = 1               # axi-symmetric co-ordinates
      icoef5 = 0               # stokes flow, neglecting convective terms v.v
      coef6 =  eps             # penalty parameter
      coef7 =  rho             # density
      coef12 =  eta            # viscosity
   elgrp1 ( nparm=10 )        # The coefficients for the mass flux bc
                              # are defined by 10 parameters
      icoef3 = 2
      icoef4 = 1              # axi-symmetric co-ordinates
      icoef5 = 2              # 5:  Degree of freedom (2=v)
      coef6  =  massflux      # 6:  Mass flux
      coef7 = 1d6
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number=1 # input for iteration 2
   elgrp2
      icoef5 = 1              # Picard's linearization
end

change coefficients, sequence_number=2 # input for iteration 3
   elgrp2
      icoef5 = 2              # Newton's linearization
end

# input for non-linear solver

nonlinear_equations             # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-3,print_level=2, lin_solver=1
   equation 1
```

```
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

end_of_sepran_input
```

Again there is no need to repeat the input file for seppost.

7.1.11.4 3D flow with given mass flux by global unknown

In order to get this example into your local directory use the command

```
sepgetex tube3d
```

This example is of course exactly the same as the previous, however, for demonstration purposes we extended it to a real 3D problem. In the 3d case it is not longer possible to give the mass flux in combination with a penalty approach, since one can not define a large surface element. For that reason we need the option with the global unknown.
In the mesh input the lower circle is subdivided into five parts in order to get nicely shaped quadrilaterals. Applying general immediately results in quadrilaterals that give bad approximations of the solution.
The rest of the input is very similar to that for the pipe in Section 2.5.2 of the Users Manual. The mesh input file is for example

```
#  tube3d.msh
#  stationary laminar Newtonian flow
#  mesh file for axi-symmetric flow of water in a tube
#
#  See Manual Examples Section 7.1.11
#
#  To run this file use:
#     sepmesh tube3d.msh
#
#  Creates the file meshoutput
#
#  Define some rectangle constants
#
constants           # See Users Manual Section 1.4
   integers
      nelmh = 3     # Number of elements along a quarter of a circle in
                    # the bottom surface
      nelmv = 4     # Number of elements in the vertical direction
                    # (pipe surface)
   reals
      radius = 20   # Radius of a circle in the bottom surface
      height = 40   # Height of the pipe
      halfr = 10    # Half the radius of a circle in the bottom surface
end
#
#  Define the mesh
#
mesh3d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1 = (0,0,0)                # centroid of circle in bottom surface
      pd2 = ( radius,0,0)         # points on circle
      pd3 = ( radius,90,0)        # pd means define in polar coordinates
      pd4 = ( radius,180,0)       # (r,phi,z), with phi in degrees
      pd5 = ( radius,270,0)
      p6  = ( radius,0, height)   # point on upper surface above p2
      pd12 = ( halfr,0,0)         # In order to create nice quadrilaterals
      pd13 = ( halfr,90,0)        # an internal square is defined with
```

```
      pd14 = ( halfr,180,0)        # end points half way the centroid and
      pd15 = ( halfr,270,0)        # points on the circle
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1 = arc2(p2,p3,p1,nelm= nelmh)   # one quarter of circle in bottom
                                        # surface
      c2 = arc2(p3,p4,p1,nelm= nelmh)   # a circle in 3D needs at least three
      c3 = arc2(p4,p5,p1,nelm= nelmh)   # sub arcs
      c4 = arc2(p5,p2,p1,nelm= nelmh)
      c5 = curves(c1,c2,c3,c4)          # Complete circle
      c6 = line2(p2,p6,nelm= nelmv)     # straight line from p2 to p6
      c7 = translate c5(p6,-p6)         # Copy of circle in bottom surface
                                        # to top surface
      c10 = line2(p12,p13,nelm= nelmh)  # straight line from p12 to p13
      c11 = line2(p13,p14,nelm= nelmh)  # straight line from p13 to p14
      c12 = line2(p14,p15,nelm= nelmh)  # straight line from p14 to p15
      c13 = line2(p15,p12,nelm= nelmh)  # straight line from p15 to p12
      c14 = line2(p12,p2,nelm= nelmh)   # straight line from p12 to p2
      c15 = line2(p13,p3,nelm= nelmh)   # straight line from p13 to p3
      c16 = line2(p14,p4,nelm= nelmh)   # straight line from p14 to p4
      c17 = line2(p15,p5,nelm= nelmh)   # straight line from p15 to p5
      c18 = curves(c10,c11,c12,c13)     # Complete square
#
#  surfaces
#
   surfaces           # See Users Manual Section 2.4
      s1 = rectangle6 (c10,c11,c12,c13 )  # subdivision of square
      s2 = rectangle6 (c1,-c15,-c10,c14 ) # subdivision of parts between
      s3 = rectangle6 (c2,-c16,-c11,c15 ) # circle and square
      s4 = rectangle6 (c3,-c17,-c12,c16 ) #
      s5 = rectangle6 (c4,-c14,-c13,c17 ) #
      s6 = surfaces(s1,s2,s3,s4,s5)       # Complete lower surface
      s7 = translate s6 (c7)              # upper surface
      s8 = pipesurface6(c5,c7,c6)         # pipe surface
#
#  volumes
#
   volumes           # See Users Manual Section 2.5
      v2 = pipe14(s6,s7,s8)             # Complete pipe
#
#   Define element groups
#
   meshvolm             # surface elements for Navier-Stokes
      velm1 = (v2)
   meshconnect          # connection elements for periodical boundary conditions
      celm2 = surfaces(s6,s7)
   plot, eyepoint = (40, 30, 50)     # make a plot of all parts
                                     # and also of the final mesh
                                     # See Users Manual Section 2.2
end
```

The input file for sepcomp is very similar to that of tubeax3:

```
# tube3d.prb
```

```
#
#  problem file for the axi-symmetric flow of water in a tube
#  stationary laminar Newtonian flow
#  penalty function approach
#  Periodical boundary conditions
#  Given mass flow with unknown constant
#  See Manual Examples Section 7.1.11
#
#  To run this file use:
#     sepcomp tube3d.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1d-6              # penalty parameter for Navier-Stokes
      rho       = 998.2d-6          # density [g/mm3]
      eta       = 1.002d-3          # viscosity [g/mm/s] [Pa.s]
      massflux  = 4000*pi           # The mass flux is equal to 4000 pi
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
      elgrp1 = 900             # type number for Navier-Stokes without swirl
      elgrp2 = -1              # type number for periodical boundary conditions
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
      surfaces (s8)            # fixed side wall
   global_unknowns             # define element group for global unknown
      glgrp1=913               # Type number for given mass flux
   global_elements
      gelm1 = surfaces(s6)   # mass flux is defined along inflow boundary
end

#  Define the structure of the large matrix

matrix                       # See Users Manual Section 3.2.4
               nosplit       # Non-symmetrical profile matrix
                             # So a direct method will be applied
                             # The matrix may not be splitted
end


#  Define the coefficients for the problems
```

```
#  See Users Manual Section 3.2.6
#  See also standard problems Section 7.1

coefficients
   elgrp1 (nparm = 20)      # coefficients for Navier-Stokes
      icoef5 = 0            # stokes flow, neglecting convective terms v.v
      coef6 =  eps          # penalty parameter
      coef7 =  rho          # density
      coef12 =  eta         # viscosity
   glgrp1 ( nparm=10 )     # The coefficients for the mass flux bc
                           # are defined by 10 parameters
      icoef5 = 3           # 5:  Degree of freedom (3=w)
      coef6  =  massflux   # 6:  Mass flux
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number=1 # input for iteration 2
   elgrp1
      icoef5 = 1           # Picard's linearization
end

change coefficients, sequence_number=2 # input for iteration 3
   elgrp1
      icoef5 = 2           # Newton's linearization
end

# input for non-linear solver

nonlinear_equations           # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-3,print_level=2, lin_solver=1//
   at_error return
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

#  Define the structure of the program

structure                     # See Users Manual Section 3.2.3
   # essential boundary conditions
   prescribe_boundary_conditions,velocity
   # compute velocity
   solve_nonlinear_system,velocity
   # compute pressure
   derivatives,pressure
   output
end

# compute the pressure as a derivative

derivatives                    # See Users Manual Section 3.2.11
```

```
   seq_input_vector1=velocity
   icheld = 7
end

end_of_sepran_input
```

To show the results of the computations it is necessary to consider cross-sections of the mesh. In this case we consider some cross-sections for constant values of z. In these cross-sections the contour of the w-velocity is plotted. The other two components are practically zero.

```
#  tube3d.pst
#  Input file for postprocessing for the axi-symmetric flow of water in a tube
#  See Manual Examples Section 7.1.11
#
#
#  To run this file use:
#     seppost tube3d.pst > tube3d.out
#
#  Reads the files meshoutput and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2
#
#  Compute cross-section with planes z = 0, 5, 10, 20 and 30
#  See Users Manual Section 5.2
#  Make a contour plot in these cross sections of the z-component
#  See Users Manual Section 5.4

   compute  cross_0 = intersection  velocity, numbunknowns=3, plane=(z=0)
   plot contour  cross_0, degfd=3
   compute  cross_5 = intersection  velocity, numbunknowns=3, plane=(z=5)
   plot contour  cross_5, degfd=3
   compute  cross_10 = intersection  velocity, numbunknowns=3, plane=(z=10)
   plot contour  cross_10, degfd=3
   compute  cross_20 = intersection  velocity, numbunknowns=3, plane=(z=20)
   plot contour  cross_20, degfd=3
   compute  cross_40 = intersection  velocity, numbunknowns=3, plane=(z=40)
   plot contour  cross_40, degfd=3

#  Compute the pressure in the symmetry plane and make a coloured contour plot

   compute  press_sym = intersection  pressure, plane(y=0)
   plot coloured levels  press_sym

end
```

The results show that the solution is constant in each cross-section.

7.1.11.5 3D flow with given mass flux by global unknown and local transforms

In order to get this example into your local directory use the command

```
sepgetex tube3dlt
```

This example is completely identical to the previous one. The same mesh is used; the main program and the post file are the same as before.
The only difference is that a local transformation is used, demonstrating the use of local transformations in 3D. The local transformation is applied to upper and lower face. Due to this option the first unknown in each of these surfaces is the normal component and the other two are the tangential components.
Since we use periodical boundary conditions it is necessary that both normal components point into the same direction and as a consequence it is necessary that one of the normals points inwardly and one outwardly. Furthermore the direction of the first tangential vector must be the same in both surfaces. To achieve this we prescribe the tangential vector by the option `tang=line(p2,p4)`. P2 and P4 are two points in the lower surface. Since the lower surface is parallel to the upper surface and the tangential vector only defines a direction, it is sufficient to use these two points even for the upper surface.
Due to the local transform and the fact that the mass flux is defined in the normal direction, it is not longer necessary to prescribe icoef5. Instead the default value 1 (normal direction is first direction) is used.
The problem input file corresponding to this case is:

```
# tube3dlt.prb
#
#  problem file for the axi-symmetric flow of water in a tube
#  stationary laminar Newtonian flow
#  penalty function approach
#  Periodical boundary conditions
#  Given mass flow with unknown constant
#  Local transformations are used
#  See Manual Examples Section 7.1.11
#
#  To run this file use:
#     tube3dlt < tube3dlt.prb
#
#  Reads the file meshoutput
#  Creates the files sepcomp.inf and sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1d-6              # penalty parameter for Navier-Stokes
      rho       = 998.2d-6          # density [g/mm3]
      eta       = 1.002d-3          # viscosity [g/mm/s] [Pa.s]
      massflux  = 4000 * pi         # The mass flux is equal to 4000 pi
   vector_names
      velocity
end
#
#  Define the type of problem to be solved
#
```

```
problem                    # See Users Manual Section 3.2.2

   types                         # Define types of elements,
      elgrp1 = 900               # type number for Navier-Stokes without swirl
      elgrp2 = -1                # type number for periodical boundary conditions
   essbouncond                   # Define where essential boundary conditions are
                                 # given (not the value)
      surfaces (s8)              # fixed side wall
   localtransform                # Define local transformations along surfaces
                                 # S6 and S7
                                 # The first unknown is the normal direction
                                 # The first tangential direction (unknown 2)
                                 # is defined by the "tang" keyword
                                 # The boundary is not transformed, since there
                                 # we have Dirichlet boundary conditions
      surfaces(s6), tang=line(p2,p4), normal = inward, skip_boundary
                                 # Along surface S6 the normal is directed
                                 # inwardly
      surfaces(s7), tang=line(p2,p4), normal = outward, skip_boundary
                                 # Along surface S7 the normal is directed
                                 # outwardly
   global_unknowns               # define element group for global unknown
      glgrp1=913                 # Type number for given mass flux
   global_elements
      gelm1 = surfaces(s6)   # mass flux is defined along inflow boundary
end

#  Define the structure of the large matrix

matrix                        # See Users Manual Section 3.2.4
               nosplit        # Non-symmetrical profile matrix
                              # So a direct method will be applied
                              # The matrix may not be splitted
end


#  Define the coefficients for the problems
#  See Users Manual Section 3.2.6
#  See also standard problems Section 7.1

coefficients
   elgrp1 (nparm = 20)        # coefficients for Navier-Stokes
      icoef5 = 0              # stokes flow, neglecting convective terms v.v
      coef6 =  eps            # penalty parameter
      coef7 =  rho            # density
      coef12 =  eta           # viscosity
   glgrp1 ( nparm=10 )       # The coefficients for the mass flux bc
                             # are defined by 10 parameters
                             # 5:  Degree of freedom (1=u_n)
                             #     Since 1 is the default this parameter does not
                             #     have to be prescribed.
      coef6  =  massflux     # 6:  Mass flux
end

#  Define the coefficients for the next iterations
```

```
#  See Users Manual Section 3.2.7

change coefficients, sequence_number=1 # input for iteration 2
   elgrp1
      icoef5 = 1           # Picard's linearization
end

change coefficients, sequence_number=2 # input for iteration 3
   elgrp1
      icoef5 = 2           # Newton's linearization
end

# input for non-linear solver

nonlinear_equations            # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-3,print_level=2, lin_solver=1//
   at_error return
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

end_of_sepran_input
```

7.1.11.6 3D flow with iterative solver

In order to get this example into your local directory use the command

```
sepgetex tube3dit
```

This example is completely identical to the example in 7.1.11.4. The same mesh is used; the main program and the post file are the same as before.
The only difference is that an iterative solver is used.
The problem input file corresponding to this case is:

```
# tube3dit.prb
#
#  problem file for the axi-symmetric flow of water in a tube
#  stationary laminar Newtonian flow
#  integrated approach, using iterative solver
#  Periodical boundary conditions
#  Given mass flow with unknown constant
#  See Manual Examples Section 7.1.11.6
#
#  To run this file use:
#     sepcomp tube3dit.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1d-6              # penalty parameter for Navier-Stokes
      rho       = 998.2d-6          # density [g/mm3]
      eta       = 1.002d-3          # viscosity [g/mm/s] [Pa.s]
      massflux  = 4000*pi           # The mass flux is equal to 4000 pi
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
      elgrp1 = 902              # type number for Navier-Stokes without swirl
      elgrp2 = -1               # type number for periodical boundary conditions
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
      surfaces (s8)             # fixed side wall
#      degfd1,degfd2, surfaces (s6)# fully developed flow (u=0)
#      degfd1,degfd2, surfaces (s8)# fully developed flow (u=0)
   global_unknowns              # define element group for global unknown
      glgrp1=913                # Type number for given mass flux
   global_elements
```

```
      gelm1 = surfaces(s6)   # mass flux is defined along inflow boundary
   renumber (1,2,3)(4,5,6,7)
end

#  Define the structure of the large matrix

matrix                       # See Users Manual Section 3.2.4
   storage_scheme = compact # Non-symmetrical compact matrix
                             # So an iterative method will be applied
end


#  Define the coefficients for the problems
#  See Users Manual Section 3.2.6
#  See also standard problems Section 7.1

coefficients
   elgrp1 (nparm = 20)       # coefficients for Navier-Stokes
      icoef5 = 0             # stokes flow, neglecting convective terms v.v
      coef6 =  eps           # penalty parameter
      coef7 =  rho           # density
      coef12 =  eta          # viscosity
   glgrp1 ( nparm=10 )      # The coefficients for the mass flux bc
                            # are defined by 10 parameters
      icoef5 = 3            # 5:  Degree of freedom (3=w)
      coef6  =  massflux    # 6:  Mass flux
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number=1 # input for iteration 2
   elgrp1
      icoef5 = 1           # Picard's linearization
end

# input for non-linear solver

nonlinear_equations             # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-3,print_level=2, lin_solver=1//
   at_error return
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
end

#  Define the structure of the program

structure                      # See Users Manual Section 3.2.3
   # essential boundary conditions
   prescribe_boundary_conditions,velocity
   # compute velocity
   solve_nonlinear_system,velocity
   # compute pressure
```

```
   derivatives,pressure
   output
end

# compute the pressure as a derivative

derivatives     # See Users Manual Section 3.2.11
   seq_input_vector1=velocity
   icheld = 7
end

#  input for the linear solver
#  See Users Manual Section 3.2.8

solve,sequence_number = 1
   iteration_method=cg, accuracy=1d-2, preconditioning = ilu, print_level=2//
   maxiter = 100, at_error resume
end

end_of_sepran_input
```

7.1.11.7 3D flow using symmetry planes

In order to get this example into your local directory use the command

```
sepgetex parttube
```

To run this example use:

```
sepmesh parttube.msh
seplink parttube
parttube < parttube.prb
seppost parttube.pst
```

The mesh can be viewed immediately after the sepmesh command and the results of the computation at the end.
This example is nearly the same as example in 7.1.11.4.
However, in this case we use only a part of the 3D region. In Figure 7.1.11.3 the boundary curves are given. The lower surface (S1) is the inflow surface, the top surface (S2) the outflow

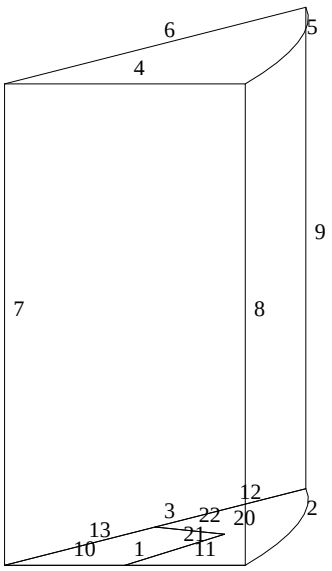


Figure 7.1.11.3: Definition of part of tube with generating curves

surface. The tube surface (S4) has noslip boundary conditions.
The front surface S3 (y=0), is a symmetry plane. There is no flow perpendicular to this surface.
Also the back surface S5 is a symmetry plane. Since this surface is not in the direction of one of the coordinate axis it is necessary to define local transformations in order to make the normal direction the first local coordinate.
The angle between S3 and S5 may be given in the input.
In this example we use tri-quadratic hexahedrons. As a consequence all surfaces must be subdivided in bi-quadratic quadrilaterals. However, the top and bottom surface are of triangular shape, which makes the creation of quadrilaterals a harder task. To that end the bottom surface is subdivided into two parts, which are subdivided by submesh generator RECTANGLE

and QUADRILATERAL respectively. The result is shown in Figure 7.1.11.4. This idea is copied from Dirk de Wachter of Ghent University. The mesh input file is

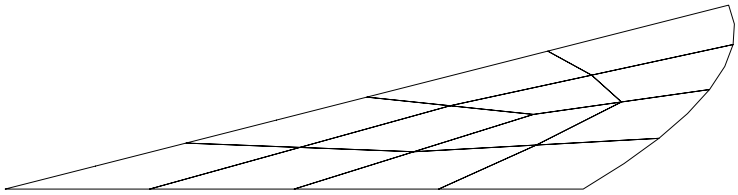


Figure 7.1.11.4: Subdivision of lower surface

```
#  parttube.msh
#  stationary laminar newtonian flow
#  mesh file for axisymmetric flow of water in a tube
#
#  See Manual Examples Section 7.1.11.7
#
#  To run this file use program parttubemesh
#     sepmesh parttube.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants
  integers
      nelmh = 4          # Number of elements along a part of a circle in
                         # the bottom surface
      nelmv = 2          # Number of elements in the vertical direction
                         # (pipe surface)
      nelmr = 2          # Number of elements in radial direction / 2
      nelmh2 = nelmh/2   # nelmh / 2
  reals
      radius = 20        # Radius of a circle in the bottom surface
      height = 40        # Height of the pipe
      angle  = 45        # angle of part of cylinder
      half_angle = angle*0.5  # angle/2
      half_radius = radius/2  # radius/2
      rad_between = 0.75*radius  # 3/4 radius
end
#
#  Define the mesh
#
mesh3d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
```

```
      p1  = (0,0,0)                    # centroid of circle in bottom surface
      pd2 = ( radius,0,0)              # points on circle
      pd3 = ( radius, angle,0)         # pd means define in polar coordinates
                                       # (r,phi,z), with phi in degrees
      p4  = (0,0, height)              # point on upper surface above p1
      pd5 = ( radius,0, height)        # point on upper surface above p2
      pd6 = ( radius, angle, height)   # point on upper surface above p3
      p11 = ( half_radius,0,0)         # point in the middle of p1,p2
      pd12 = ( rad_between, half_angle,0) # special point to define extra
                                       # quadrilateral
      pd13 = ( half_radius, angle,0)   # point in the middle of p1,p3
#
#  curves
#
   curves           # See Users Manual Section 2.3
      c1  = curves(c10, c11)         # Line from p1 to p2, splitted into 2 parts
      c10 = line2 ( p1, p11, nelm =  nelmr )
      c11 = line2 ( p11, p2, nelm =  nelmr )
      c2  = arc2(p2,p3,p1,nelm= nelmh)   # circle part in bottom surface
      c3  = curves(c12, c13)         # Line from p3 to p1, splitted into 2 parts
      c12 = line2 ( p3, p13, nelm =  nelmr )
      c13 = line2 ( p13, p1, nelm =  nelmr )
      c4  = translate c1 (p4,-p5)    # Line in top surface
      c5  = translate c2 (p5,p6)     # Line in top surface
      c6  = translate c3 (p6,-p4)    # Line in top surface
      c7  = line2 ( p1, p4, nelm =  nelmv ) # generating line from bottom to top
      c8  = line2 ( p2, p5, nelm =  nelmv ) # generating line from bottom to top
      c9  = line2 ( p3, p6, nelm =  nelmv ) # generating line from bottom to top
      c20 = curves(c21, c22)         # Extra line from p11 to p13,
                                     # splitted into 2 parts
      c21 = line2 ( p11, p12, nelm =  nelmh2 )
      c22 = line2 ( p12, p13, nelm =  nelmh2 )
#
#  surfaces
#
   surfaces           # See Users Manual Section 2.4
      s1  = surfaces(s11,s12)            # bottom surface splitted into 2
                                         # surfaces to make it simple to create
                                         # quadrilaterals
      s11 = quadrilateral6(c10,c21,c22,c13)
      s12 = rectangle6(c11,c2,c12,-c20)
      s2  = translate s1 (c4,c5,c6)      # top surface
      s3  = pipesurface6(c1,c4,c7,c8)    # First pipe surface
      s4  = pipesurface6(c2,c5,c8,c9)    # Outer pipe surface
      s5  = pipesurface6(c3,c6,c9,c7)    # Last pipe surface
      s6  = ordered surface(s3,s4,s5)    # Complete pipe surface
#
#  volumes
#
   volumes           # See Users Manual Section 2.5
      v1=pipe14(s1,s2,s6)
#
#   Define element groups
#
   meshsurf         # surface elements on top surface
```

```
      selm1 = s2
   meshvolume       # volume elements
      velm2 = v1

   plot, eyepoint=(40,30,50)

end
```

Figure 7.1.11.5 shows the final mesh.

In this example we have introduced an extra item. The pressure is given at the outflow so we

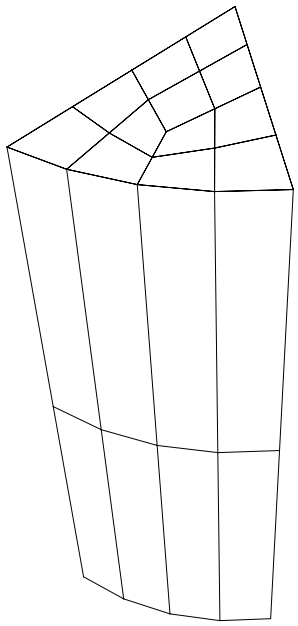


Figure 7.1.11.5: Hidden line plot of the final mesh

need a boundary element of type 910. However, since we must apply local transformations on the back surface, these transformations are also applied to all curves corresponding to that surface. To avoid that we might skip all the boundaries, using `skip_boundary`. This is no problem for the symmetry curve, the curve at the lower surface, nor the curve at the pipe surface. However, if we skip the top curve (C6), we are not able to apply the symmetry condition anymore. Hence we need to include curve C6. Since C6 is also part of the upper surface, this means that we have to apply local transformations to a part of the upper surface. Here we enter a typical SEPRAN problem: local transformations are not applied to boundary elements, but only to standard elements created in the mesh. Hence the surface elements must already be created in the mesh part. For that reason we have two element groups in the mesh input file.
The other problem in this example is that we want to prescribe the pressure. If we do not prescribe it, the pressure will be of the order 10^{-3}. Since the parameter ϵ in the penalty function method is related to the magnitude of the pressure, we have decided to use the integrated approach. This avoids the problem of looking for a good choice for ϵ. Now we can prescribe the pressure using boundary elements of type 910.
However, experiments show that the pressure can not be chosen with an arbitrary size. If

we increase the pressure we see that with this coarse mesh, the result is good as long as the pressure at outflow is at most of the order of 10^2. A value of 10^3 results in inaccurate computations. This is an immediate result of the inaccuracy of the velocity which also influences the accuracy of the pressure.

Since the inflow boundary conditions depend on the space coordinates, we need a function subroutine FUNCBC, and hence a main program:

```
      program parttube

!     --- Main program for axi-symmetric flow of water in a tube
!         To link this program use:
!
!         seplink parttube

      implicit none
      call sepcom (0)
      end

!     --- Define the velocity at inflow (quadratic profile)

      function funcbc ( ichois, x, y, z )
      implicit none
      integer ichois
      double precision x, y, z, funcbc, radius, r
      radius = 20d0
      r = sqrt(x**2+y**2)

      if ( ichois==1 ) then

         funcbc = 20d0 * ( 1d0-r**2/radius**2)

      end if

      end
```

The problem input file corresponding to this case is:

```
# parttube.prb
#
#  problem file for the axisymmetric flow of water in a tube
#  stationary laminar newtonian flow
#  penalty function approach
#  See Manual Examples Section 7.1.11.7
#
#  To run this file use:
#     sepcomp parttube.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
```

```
   reals
      eps       = 1d-10           # penalty parameter for Navier-Stokes
      rho       = 998.2d-6        # density [g/mm3]
      eta       = 1.002d-3        # viscosity [g/mm/s] [Pa.s]
      Pres      = -1d2            # Pressure at outflow
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
      elgrp1 = 910              # type number for boundary element for
                                # Navier-Stokes without swirl
                                # Is used to define the outflow pressure
      elgrp2 = 902              # type number for Navier-Stokes
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
      surfaces (s1)             # inflow surface
      surfaces (s4)             # fixed wall (outer pipe)
      degfd2, surfaces (s3)     # symmetry face
      degfd1, surfaces (s5)     # symmetry face, normal component
   localtransform               # define where local transformations must be
                                # applied
      surfaces(s5), transformation=standard,//             # skewed surface
      tang=line(p1,p3), normal=outward, include_curve(c6)  # include upper curve
   renumber levels (1,2,3)(4,5,6,7)   # Renumbering is necessary to avoid zero
                                # diagonal elements
end

#  Define the structure of the program

structure                     # See Users Manual Section 3.2.3
   # essential boundary conditions
   prescribe_boundary_conditions,velocity
   # compute velocity
   solve_nonlinear_system,velocity
   # compute pressure
   derivatives,pressure
   output
end

#  Fill the non-zero values of the essential boundary conditions
#  See Users Manual Section 3.2.5

essential boundary conditions
   surfaces(s1), degfd3, func = 1   # w is given by a function
end

#  Define the coefficients for the problems
#  See Users Manual Section 3.2.6
#  See also standard problems Section 7.1
```

```
coefficients
   elgrp1 (nparm = 15)      # coefficients for the given pressure
      coef8 =  Pres         # given pressure
   elgrp2 (nparm = 20)      # coefficients for Navier-Stokes
      icoef3 = 1
      icoef5 = 0            # stokes flow, neglecting convective terms v.v
      coef6 =  eps          # penalty parameter
      coef7 =  rho          # density
      coef12 =  eta         # viscosity
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number=1 # input for iteration 2
   elgrp1
      icoef5 = 1           # Picard's linearization
end

change coefficients, sequence_number=2 # input for iteration 3
   elgrp1
      icoef5 = 2           # Newton's linearization
end

# input for non-linear solver

nonlinear_equations              # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-3,print_level=2, lin_solver=1//
   at_error return
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

# compute the pressure as a derivative

derivatives                          # See Users Manual Section 3.2.11
   seq_input_vector1=%velocity
   skip_element_groups = 1           # skip the boundary element
   icheld = 7
end

end_of_sepran_input
```

To perform post processing, the following input file may be used

```
#  parttube.pst
#  Input file for postprocessing for the axi-symmetric flow of water in a tube
#  See Manual Examples Section 7.1.11.7
#
#
#  To run this file use:
#     seppost parttube.pst > parttube.out
```

```
#
#  Reads the files meshoutput and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2

#  Plot the results
#  See Users Manual Section 5.4

   compute vel_0 = intersection velocity, plane(z=0), numbunknown=3//
      transformation = cartesian
   plot vector vel_0
   plot coloured levels vel_0, degfd3
   compute vel_1 = intersection velocity, plane(z=20), numbunknown=3//
      transformation = cartesian
   plot vector vel_1
   plot coloured levels vel_1, degfd3
   compute vel_2 = intersection velocity, plane(z=40), numbunknown=3//
      transformation = cartesian
   plot vector vel_2
   plot coloured levels vel_2, degfd3
   compute press_sym = intersection pressure, plane(x-2y=0)
   plot coloured levels press_sym

end
```

7.1.12 A selection of examples of flow problems

In this section we supply a number of examples of flow problems that do not add some extra possibilities itself, but are nice to be used as starting point for computations. The corresponding files itself are not printed in this manual. However, you can get them easily into your local directory using the command sepgetex.
The following examples are available:

cross_vel Incompressible non-newtonian channel flow in a cross section using velocity boundary conditions.

cross_pres Incompressible non-newtonian channel flow in a cross section using pressure boundary conditions.

cylinderinst Incompressible time dependent flow of fluid cylinder in another fluid using surface tension at the interface (2D Cartesian coordinates)

sphereinst Incompressible time dependent flow of fluid particle in another fluid using surface tension at the interface (Axi-symmetric coordinates)

7.1.12.1 Example cross_vel

This problem has been provided by Juan Luis Cormenzana Carpio of the university of Madrid.
In this problem we consider the non-Newtonian flow in a channel in a cross-configuration. In the first part we prescribe parabolic velocity profiles in the inlets.
To get this example locally use the command:

```
sepgetex cross_vel
```

Figure 7.1.12.1 shows the computed velocity vectors Figure 7.1.12.2 shows the coloured pressure

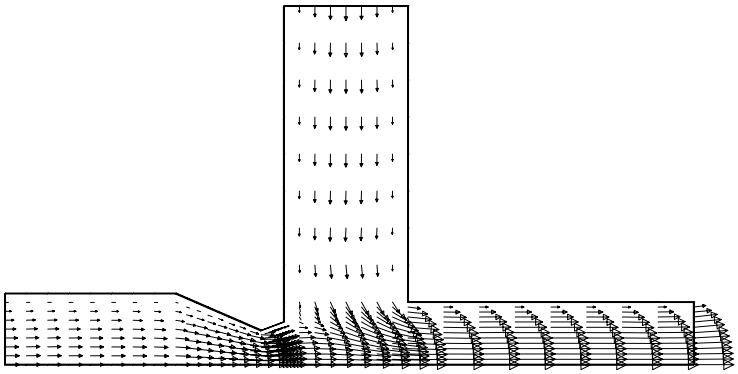


Figure 7.1.12.1: Velocity vectors for example cross_vel

levels. Figure 7.1.12.3 shows the stream lines of the computation.

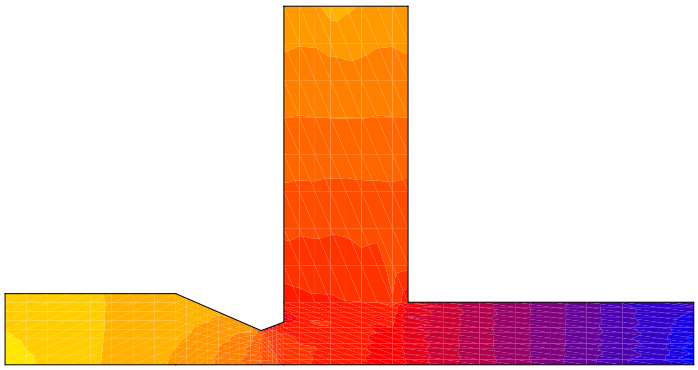


Figure 7.1.12.2: Coloured pressure levels for example cross_vel

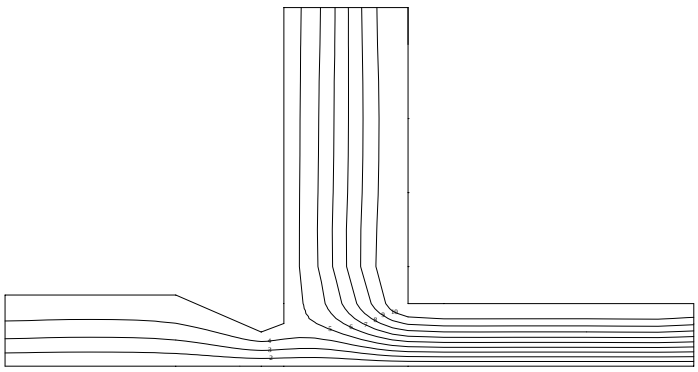


Figure 7.1.12.3: Stream lines for example cross_vel

7.1.12.2 Example cross_pres

This problem also provided by Juan Luis Cormenzana Carpio is exactly the same as the previous one. The only difference is that the pressures computed in the previous example at the inlets are used as pressure boundary conditions. So we get a minor difference in the pictures. Also in this case the convergence behaviour of the non-linear iteration process is rather strange. The following part of the output of sepcomp shows the convergence:

```
Information about the iteration process
Iteration  Equation    ||u(n)-u(n-1)|| conv. speed
     1         1          3.33E-06      1.00E+00
     2         1          1.69E-03      5.07E+02
     3         1          3.63E-02      2.15E+01
     4         1          1.37E-01      3.77E+00
     5         1          1.81E-01      1.33E+00
     6         1          1.31E-01      7.20E-01
     7         1          6.79E-02      5.19E-01
     8         1          3.02E-02      4.45E-01
     9         1          1.26E-02      4.16E-01
    10         1          5.08E-03      4.05E-01
    11         1          2.03E-03      4.00E-01
    12         1          8.09E-04      3.98E-01
    13         1          3.22E-04      3.98E-01
    14         1          1.28E-04      3.97E-01
    15         1          5.08E-05      3.98E-01
Convergence has been reached after  15 steps
```

What we see is that in the first steps very little happens and it looks as if the process is ready after one step. If we do not take precautions the program stops with the message that divergence has been found after 3 or 4 steps since the difference between succeeding iterations increases.
To prevent this message it is necessary to do at least 5 iteration before checking the convergence. In the input file this has been done by using

```
miniter = 10
```

To get this example locally use the command:

```
sepgetex cross_pres
```

Figure 7.1.12.4 shows the coloured pressure levels. Figure 7.1.12.5 shows the coloured stream function levels.

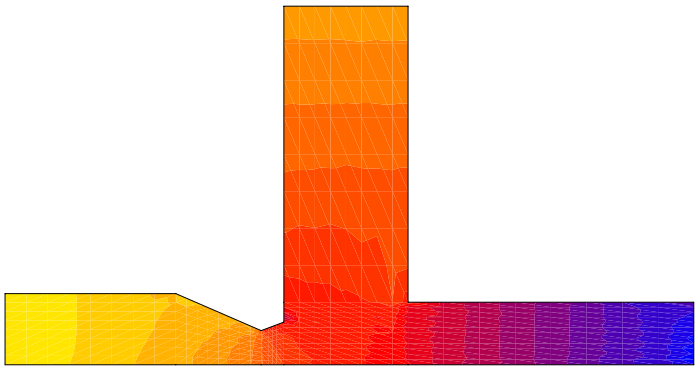


Figure 7.1.12.4: Coloured pressure levels for example cross_pres

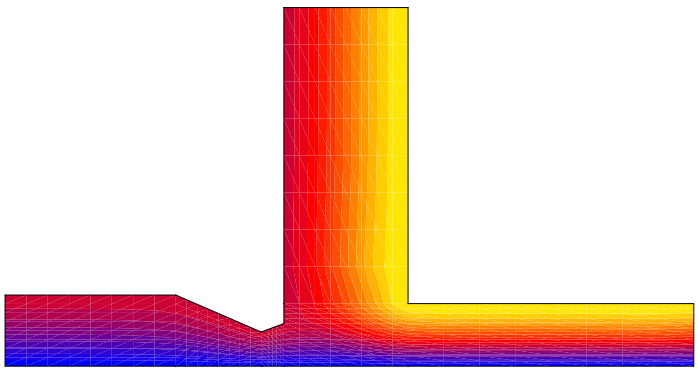


Figure 7.1.12.5: Coloured stream function levels for example cross_pres

7.1.12.3 Example cylinderinst

This problem we consider the flow of liquid cylinder in another fluid. The surrounding fluid has an upwards velocity, simulating the falling of the cylinder into this fluid. The properties of both fluids differ strongly with respect to viscosity and density. At the interface of both fluids we assume that the surface tension is present in order to prevent the solution of the cylinder in the surrounding fluid. The surface tension creates a pressure discontinuity at the interface.
In this example gravity force is introduced in order to make it possible that the cylinder falls downwards. Since there is no balance between the upwards-directed velocity of the surrounding fluid and the downwards-directed velocity of the cylinder no stationary state is reached.

To get this example locally use the command:

```
sepgetex cylinderinst
```

Figure 7.1.12.6 shows the configuration.

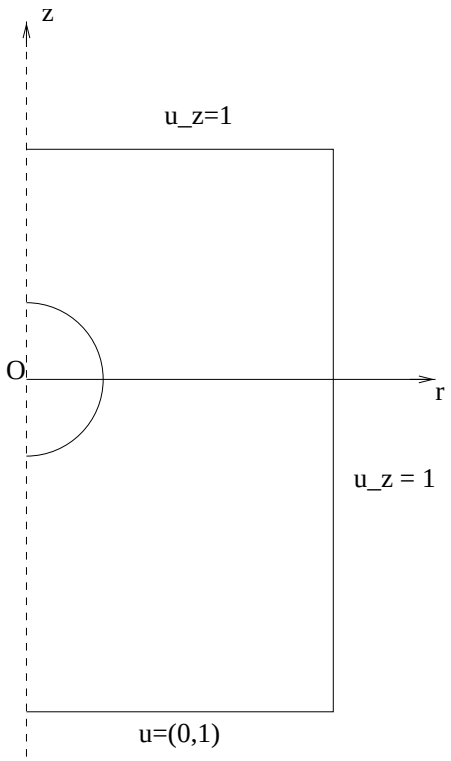


Figure 7.1.12.6: Region of definition for cylinder falling down in surrounding fluid

7.1.12.4 Example sphereinst

This problem we consider the flow of liquid particle in another fluid. The surrounding fluid has an upwards velocity, simulating the falling of the particle into this fluid. The properties of both fluids differ strongly with respect to viscosity and density. In fact this example is exactly the same as for the instationary cylinder, with the exception that the flow is axi-symmetric instead of Cartesian.

To get this example locally use the command:

```
sepgetex sphereinst
```

Since the subroutine for the surface tension has only been implemented for Cartesian coordinates it is necessary to translate the surface tension into a given normal stress.
This normal stress is defined by $\sigma_n = \gamma(\frac{1}{R_1} + \frac{1}{R_2})$, and since for the sphere we have $R_1 = R_2$ is the radius of the sphere, we can translate this into $\sigma = -\mathbf{n}\gamma(\frac{2}{R})$.
Using (0,0) as centre of the sphere $\mathbf{n}$ can be written as $\mathbf{n} = (\frac{\mathbf{x}}{\mathbf{R}}, \frac{\mathbf{y}}{\mathbf{R}})$
Because the stress depends on space a function subroutine FUNCCF is required and hence a main program must be provided. This program looks like:

```
      program sphereinst
      call sepcom ( 0 )
      end

!     --- funccf is used to define the stress along the interface
!         This stress is defined by the surface tension
!         For 3D the surface tension is defined as
!
!         sigma = - gamma ( 1/R_1 + 1/R_2 ) n
!
!         with n the outward directed normal
!
!         The outwards directed normal is defined as
!
!         n = ( x/R, y/R )
!
!         and for a sphere we have R_1 = R_2 = R
!
      function funccf(ichoice,x,y,z)
      implicit none
      integer ichoice
      double precision funccf,x,y,z

!     --- get gamma and the radius from the input file

      double precision radius, gamma, getconst
      save  radius, gamma
      integer ifirst
      data ifirst /0/
      if ( ifirst==0 ) then

!     --- ifirst = 0, first call
!         get gamma and radius from input file
!         Since they are saved, this has to be done only once

         gamma = getconst ( 'gamma' )
         radius = getconst ( 'radius' )
```

```
         ifirst = 1  ! make sure that this part is done only once

      end if  ! ( ifirst==0 )

      if ( ichoice==1 ) then

!     --- x-component of stress

         funccf = -2d0*gamma*x/radius**2

      else

!     --- y-component of stress

         funccf = -2d0*gamma*y/radius**2

      end if
      end
```

7.1.13 Computation of Drag Coefficients of a Sphere

In this section we show how the drag coefficients of a sphere may be computed. This example is inspired by the report of Tabata and Itakura (1995), who defined a kind of benchmark for the computation of drag coefficients.

In order to get this example into your local directory use the command

```
sepgetex drag
```

Let G be a body in a velocity field. Let U be the representative velocity and ρ be the density of the fluid. The drag coefficient of G is defined by:

$$C_D = \frac{D}{\frac{1}{2}\rho U^2 A}, \tag{7.1.13.1}$$

where D is the total force exerted on G by the fluid and A is the area of the cross section of G in the direction U.
In this example we consider a sphere. It is sufficient to reduce the problem to a two-dimensional axi-symmetric one, with a symmetry axis subdividing the sphere into two equal parts.
The uniform velocity is chosen from below and has been normalized to $U_z = 1$. Figure 7.1.13.1 shows the configuration. On the symmetry-axis we use the symmetry condition $u_r = 0, \tau_{rz} = 0$.

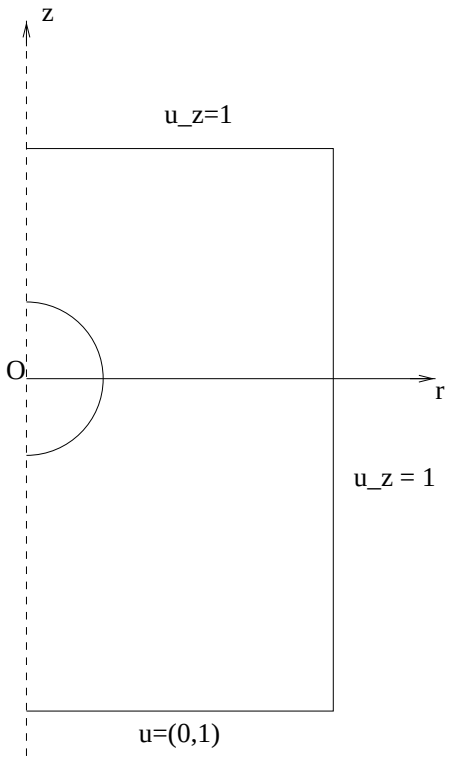


Figure 7.1.13.1: Region of definition for flow around sphere

On the inflow boundary we impose the uniform velocity $\mathbf{u} = (\mathbf{0}, \mathbf{1})$.
On the other boundaries we use $u_z = 1$ and $\tau_{zz} = 0$.
The area A is given by $A = \pi R^2$, where R is the radius of the sphere.
The Reynolds number is defined as $RE = \frac{\rho LU}{\mu}$, with U the uniform velocity, L the diameter of the sphere and μ the viscosity.
In the report of Tabata and Itakura tables are given of the drag coefficient for various values of the Reynolds number. We have computed several of these values for Reynolds ranging from 10 to 200

and found a very good agreement of at least 3 digits.
The mesh chosen is identical to that of Tabata and Itakura. Figure 7.1.13.2 shows the curves that are used. In order to create the program sepmesh has been used with the following input file:

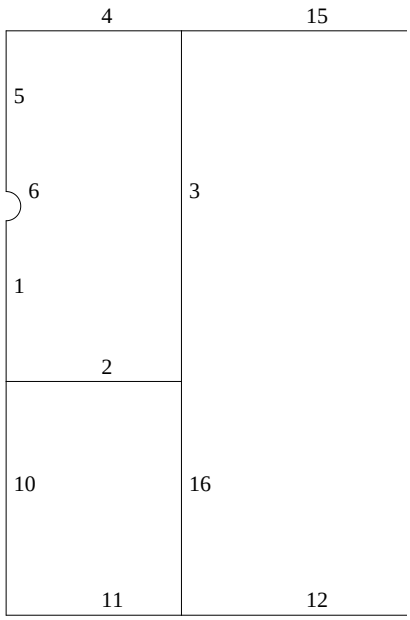


Figure 7.1.13.2: Curves defining the region around the sphere

```
#   drag.msh
#
#   Mesh for flow round a fixed sphere
#   See Manual Examples Section 7.1.13
#
#   This example is used to compute the drag coefficient of a sphere
#   The mesh used is the one shown in:
#   Masahisa Tabata and Kazuhiro Itakura
#   Pecise Computation of Drag Coefficients of the sphere
#   Department of Mathematics, Hiroshima University, Japan
#
#   The problem is solved using axi-symmetric coordinates, which implies that
#   the sphere reduces to a half circle in the (r,z)-plane
#
#
#   Following Tabata, the region is dubdivided into 3 separate regions:
#
#
#   ----------------------------
#   |                |           |
#   |                |           |
#   |                |           |
#   |                |           |
#   |                |           |
```

```
#   |                |           |
#   \       I        |           |
#    \               |           |
#    /               |   |||     |
#   /                |           |
#   |                |           |
#   |                |           |
#   ------------------           |
#   |                |           |
#   |      ||        |           |
#   |                |           |
#   ------------------------------
#
#  To create the mesh run:
#
#  sepmesh sphere1.msh
#
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants
    reals
       radius =  0.5        # Radius of sphere
       z1        = 14       # Height of the outer region below the
                            # Centre of the sphere
       z2        =  6       # Height of the outer flow region above the
                            # centre of the sphere
       rr        = 14       # width of the outer flow region
       fact      = 1.2      # Each next element along symmetry axis has length
                            # of previous one times fact
    integers
       lin       = 2        # Quadratic line elements
       sur       = 4        # Quadratic surface elements
       n_hor     = 4        # number of elements in horizontal direction of
                            # the square around the sphere
       n_ver     = 8        # number of elements in vertical direction of
                            # the square around the sphere
       n_circ    =16        # Number of elements along the sphere
end
#
#  Define the mesh
#
mesh2d            # See Users Manual Section 2.2
#
#  user points
#
points             # See Users Manual Section 2.2
      p 1=(0,0)               # Center of sphere
      p 2=(0,- radius)        # lowest point of sphere
      p 3=(0,- z1)            # Point left under of outer region
      p 4=( rr,- z1)          # Point right under of outer region
      p 6=( rr, z2)           # Point right upper of outer region
      p 7=(0, z2)             # Point left upper of outer region
```

```
      p 8=(0, radius)          # highest point of sphere
      p10=(0,- z2)             # under left point of inner square
      p11=( z2,- z2)           # under right point of inner square
      p12=( z2, z2)            # upper right point of inner square
      p13=( z2,- z1)           # extra point on lower boundary
      p14=( rr,- z2)           # extra point on right-hand side boundary
#
#   curves
#
   curves          # See Users Manual Section 2.3

   #  First the region around the sphere

      c1=line  lin (p2,p10,nelm= n_circ,ratio=2,factor= fact)
                        # Lower part of symmetry axis in square
      c2=line  lin (p10,p11,nelm= n_hor)
                        # Lower boundary of square
      c3=line  lin (p11,p12,nelm= n_ver)
                        # right-hand-side boundary of square
      c4=line  lin (p12,p7,nelm= n_hor)
                        # Upper boundary of square
      c5=line  lin (p7,p8,nelm= n_circ,ratio=4,factor= fact)
                        # Upper part of symmetry axis in square
      c6=arc  lin (p8,p2,-p1,nelm= n_circ)
                        # face of sphere
      c7 = curves(c2,c3,c4)

      # Next the curves for the outer region

      c10=line  lin (p10,p3,nelm= n_hor)
                        # Lower part of symmetry axis in outer region
      c11=line  lin (p3,p13,nelm= n_hor)
                        # Left-hand part of lower boundary in outer region
      c12=line  lin (p13,p4,nelm= n_hor)
                        # Right-hand part of lower boundary in outer region
      c13=line  lin (p4,p14,nelm= n_hor)
                        # Lower part of right-hand boundary in outer region
      c14=line  lin (p14,p6,nelm= n_ver)
                        # Upper part of right-hand boundary in outer region
      c15=line  lin (p6,p12,nelm= n_hor)
                        # Upper boundary in outer region
      c16=line  lin (p11,p13,nelm= n_hor)
                        # Extra line in outer region
      c17 = curves(c13,c14)   #  right-hand boundary in outer region
      c18 = curves(-c3,c16)   #  left-hand boundary in outer region
      c30 = curves(c11,c12)   #  inflow boundary
      c31 = curves(c15,c4 )   #  outflow boundary
      c32 = curves(c1,c10)    #  lower part symmetry axis
 #
 #  Define the surfaces
 #
   surfaces       # See Users Manual Section 2.4
      s1=rectangle  sur (c1,c7,c5,c6)       # Square
      s2=rectangle  sur (c10,c11,-c16,-c2)  # Lower part of outer region
      s3=rectangle  sur (c12,c17,c15,c18)   # Right-hand part of outer region
```

```
   plot            # make a plot of the mesh
                   # See Users Manual Section 2.2

end
```

The reason to choose such a mesh is just to get exactly the same mesh as in the report. Figure 7.1.13.3 shows the mesh created. In order to compute the drag coefficient it is necessary to compute

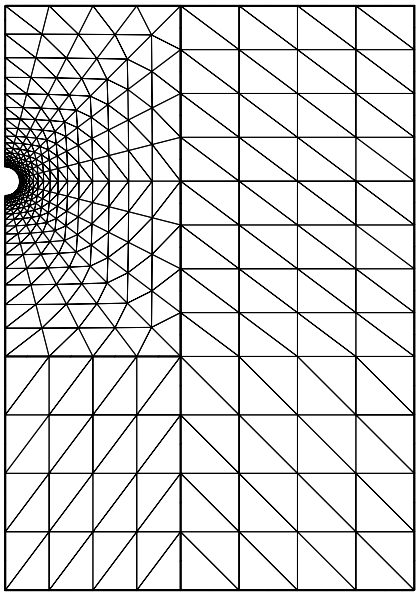


Figure 7.1.13.3: mesh for flow around the sphere

the forces that are exerted on the sphere. The easiest way to do so is to use reaction forces. These forces must be added in order to get the integral of the force, since the reaction forces already consists of integrals.
The computation of the drag coefficient requires the evaluation of a coefficient that must be multiplied by the drag.

sepcomp requires input, which is given in the following part:

```
#   drag.prb
#
#   problem file for flow round a fixed sphere
#   See Manual Examples Section 7.1.13
#
#   This example is used to compute the drag coefficient of a sphere
#   The mesh used is the one shown in:
#   Masahisa Tabata and Kazuhiro Itakura
#   Precise Computation of Drag Coefficients of the sphere
#   Department of Mathematics, Hiroshima University, Japan
#
#   The problem is solved using axi-symmetric coordinates, which implies that
```

```
#   the sphere reduces to a half circle in the (r,z)-plane
#
#   The penalty function method is used to solve the Navier-Stokes equations
#
# Define some general constants
#
set warn off   ! suppress warnings
constants      # See Users Manual Section 1.4
   reals
      eps    = 1d-6           # penalty parameter for Navier-Stokes
      rho    = 1              # density in flow
      eta    = 0.1            # viscosity in flow (Re=10)
      veloc  = 1              # Uniform z-velocity of flow
      radius = 0.5            # Radius of sphere (used to compute the drag
                              # coefficient)
   integers
      cur_sphere  = 6         # sphere surface
      cur_in      = 30        # Inflow boundary
      cur_out     = 31        # Outflow boundary
      cur_rhs     = 17        # Right-hand-side boundary
      cur_sym1    = 32        # symmetry axis lower part
      cur_sym2    = 5         # symmetry axis upper part
   vector_names
      velocity
      pressure
      stress
      reaction_force
   variables
      cD            # Drag coefficient
      D             # First component of integrated reaction force
      R2            # Second component of integrated reaction force
end
#
#   Define the type of problem to be solved
#
problem               #See Users Manual Section 3.2.2

   types                    # Define type of elements
                            #See Users Manual Section 3.2.2
   elgrp1=900               # Type number for Navier-Stokes, without swirl

   # Define where essential boundary conditions are present

   essbouncond
      degfd1,degfd2,curves(c cur_sphere) # velocity on sphere surface
      degfd1,curves(c cur_sym1)          # Symmetry axis (u_r = 0)
      degfd1,curves(c cur_sym2)          # Symmetry axis (u_r = 0)
      curves(c cur_in)                   # Inflow boundary, uniform flow
      degfd1,curves(c cur_out)           # Outflow boundary, u_r = 0
      curves(c cur_rhs)                  # Right-hand-side boundary,
                                         # uniform flow

   end

# Define the structure of a large matrix
```

```
matrix                       # See Users Manual Section 3.2.4
   reaction_force           # Non-symmetrical profile matrix
                             # So a direct method will be applied
                             # reaction forces are computed,
                             # these are used to compute the Drag coefficient
end

# Define the structure of the problem
# In this part it is described how the problem must be solved
#
structure                   # See Users Manual Section 3.2.3

   # Create start vector and put boundary conditions in this vector
   create_vector, sequence_number=1, velocity

   # Compute the velocity and the reaction force
   solve_nonlinear_system,sequence_number=1, velocity//
      reaction_force = %reaction_force

   # Compute the pressure
   derivatives, seq_deriv=1,seq_coef = 1, pressure

   # Compute the stress tensor
   derivatives,seq_deriv=2,seq_coef = 1, stress

   # Since the reaction force already consists of terms evaluated as
   # an integral it is sufficient to add all terms along the boundary
   # The z-component produces the drag

   boundary_integral, reaction_force, scalar1 = D, scalar2 = R2
   print D, text = 'z-component integral of reaction_force'

   # To get the Drag coefficient we must multiply by a factor.
   # This is done in the main program in function subroutine funcscal

   cD = -D / (0.5d0*radius^2*rho*veloc^2*pi)
   print cD

   # Prepare output for seppost
   output
end
#
# Boundary integrals
#
boundary_integral     # See users manual, Section 3.2.14
   ichint = 8              # Summation of fz along the boundary
   ichfun = 0              # f = 1 (default)
   degree_of_freedom  = 2  # Only the z-component is required
   curves(c cur_sphere)    # Boundary integral on sphere surface
end

# Create start vector and put the essential boundary conditions into this
# vector
```

```
create vector         # See Users Manual, Section 3.2.10
    curves(c cur_in), degfd2, value =  veloc  # Uniform flow at instream
    curves(c cur_rhs),degfd2, value =  veloc  # Uniform flow at right-hand-side
                                              # the other values are zero
end

#  Define coefficients for the problems
#  See Users Manual Section 3.2.6

coefficients

   elgrp1 (nparm=20)      # The coefficients for Navier-Stokes are defined
                          # by 20 paramerets
                          # Definition for sphere
     icoef2 = 1           # 2: type of constitutive equation (1=Newton)

     icoef4 = 1           # 4: Axi-Symmetric co-ordinates
     icoef5 = 0           # 5: Type of linearization (0=Stokes flow)
     coef6  =  eps        # 6: Penalty function parameter eps
     coef7  =  rho        # 7: Density in fluid
                          # 8: angular velocity = 0
                          # 9: body force in x-direction = 0
                          #10: body force in y-direction = 0
     coef12 =  eta        #12: Viscosity in fluid
end

#  Define the coefficients for the next iterations
#  See User Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 3:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 3:  Type of linearization (2=Newton iteration)
end

# Define the parameters for the non-linear solver

nonlinear_equations   # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-4,print_level=2, lin_solver=1//
                   at_error=return
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

# The pressure is computed as a derived quantity of the Navier-Stokes
# equation
# See Users Manual Section 3.2.11
```

```
derivatives, sequence_number = 1
   icheld=7   # pressure
end

# The stress in the same way

derivatives, sequence_number = 2
   icheld = 6         #stress
end

end_of_sepran_input
```

7.1.14 Channel flow using the gravity force as driving force

In this section we consider a simple channel flow, where we explicitly prescribe the gravity force.

In order to get this example into your local directory use the command

```
sepgetex gravity
```

The example itself is trivial, it concerns a straight channel as the one in Figure 7.1.8.1. At the inflow boundary we prescribe a uniform velocity and at the lower boundary we define free-slip boundary conditions. At the upper boundary a stress-free boundary condition is given, and at the outflow boundary the tangential stress is zero and the normal stress is given.
For this flow the velocity is uniform $\mathbf{v} = (\mathbf{1}, \mathbf{0})$.
Special in this example is the we have besides the inflow, also the gravity as driving force. This means that the pressure is not constant but depends on the height y: $p = \rho g(1 - y)$.
As a consequence the normal stress at the outflow is not longer zero, since $\sigma^{nn} = -p + \frac{\partial v}{\partial n} = -p$.
So we have to give the outflow boundary condition as a function.

The mesh file for this example is quite trivial

```
#  gravity.msh
#
#  mesh file for 2d free surface problem with gravity
#  See Manual Examples Section 7.1.14
#
#  To run this file use:
#     sepmesh gravity.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      x_left = -4
      x_right= 10
      y_top  =  1
      y_bottom = 0
   integers
      n = 6            # number of elements in length direction
      m = 4            # number of elements in width direction
      lin = 2          # quadratic elements
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=( x_left, y_bottom)      # Left under point
      p2=( x_right, y_bottom)     # Right under point
      p3=( x_right, y_top)        # Right upper point
      p4=( x_left, y_top)         # Left upper point
#
```

```
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line  lin (p1,p2,nelm= n)     # lower wall
      c2=line  lin (p2,p3,nelm= m)     # outflow boundary
      c3=line  lin (p3,p4,nelm= n)     # upper side (free surface)
      c4=line  lin (p4,p1,nelm= m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
                    # Quadratic triangles are used
      s1=rectangle4(c1,c2,c3,c4)

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2

end
```

The main program used is

```
      program gravity
      implicit none
      call sepcom ( 0 )
      end

!     --- Function funccf is used to define variable coefficients
!         in this case it concerns the value of the pressure
!         at the outflow boundary
!         Mark that g has a negative sign
!         The pressure is defined by rho g (1-y)

      function funccf(ichois,x,y,z)
      implicit none
      integer ichois
      double precision funccf,x,y,z

!     --- use common cuscons to get the values of the real constants
!         as defined in the "constants" input block

      include 'SPcommon/comcons1'
      include 'SPcommon/cuscons'

      double precision rho, g
      rho = rlcons(2)
      g = rlcons(4)
      funccf=rho*g*(1-y)
      end
```

The corresponding input file for program cavity is:

```
# gravity.prb
#
#  problem file for 2d free surface problem with gravity
```

```
#  penalty function approach
#  problem is stationary and non-linear
#  See Manual Examples Section 7.1.14
#
#  To run this file use:
#     sepcomp gravity.prb
#
#  Reads the file meshoutput
#  Creates the files sepcomp.inf and sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1d-6              # penalty parameter for Navier-Stokes
      rho       = 1                 # density
      eta       = 0.01              # viscosity
      g         = -5.4              # value of the gravity (negative sign!)
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=900                # Type number for Navier-Stokes, without swirl
                                # See Standard problems Section 7.1
   natbouncond                  # Define natural boundary conditions
      bngrp1=910                # Type number for natuaral boundary conditions
   bounelements                 # Define where natural boundary conditions
                                # are given
      belm1 = curves c2 (shape=2)       # boundary elements at outflow boundary
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      degfd2=curves(c1)         # under wall free slip
      curves(c4)                # inflow
                                # All not prescribed boundary conditions
                                # satisfy corresponding stress is zero
end
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because the integral of the pressure over the boundary
#  is required
#
structure                    # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions, velocity
   solve_nonlinear_system, velocity
```

```
   print velocity
 # Compute the pressure
   derivatives, pressure
 # Write the results to a file
   output
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c4), degfd1, value=1     # The u-component of the velocity at
                                   # instream is constant
                                   # The rest of the vector is 0
end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef6  =  eps        # 6:  Penalty function parameter eps
      coef7  =  rho        # 7:  Density
      coef10 =  g          #10:  Value of f2 (mark this is -gravity)
      coef12 =  eta        #12:  Value of eta (viscosity)
   bngrp1 ( nparm=15 )     # The coefficients for the boundary conditions
                           # are defined by 15 parameters
      coef6  = func=1      # In this case we have sigma_xx = -p given
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1//
   at_error return
   equation 1
```

```
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

# compute pressure
# See Users Manual, Section 3.2.11

derivatives, sequence_number = 1
   icheld=7          # icheld=7, pressure in nodes
                     # See Standard problems Section 7.1
end

end_of_sepran_input
```

Postprocessing may for example be applied using the input file:

```
#  gravity.pst
#  Input file for postprocessing for 2d free surface problem with gravity
#  See Manual Examples Section 7.1.8
#
#
#  To run this file use:
#     seppost gravity.pst > gravity.out
#
#  Reads the files meshoutput, sepcomp.inf and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2
#
#  Define the names of the solution vectors
#  See Users Manual Section 5.2
#
#
#  compute the stream function
#  See Users Manual Section 5.2
#  store in stream_function, and give vector a name

   compute stream_function = stream function velocity

#  Plot the results
#  See Users Manual Section 5.4

   plot vector velocity                  # Vector plot of velocity
   plot contour pressure                 # Contour plot of pressure
   plot coloured contour pressure        # Coloured contour plot of pressure
   plot contour stream_function          # Contour plot of stream function
   plot coloured contour stream_function  # Coloured contour plot of
                                            # stream function

end
```

7.1.15 A slipping fault in between two viscous fluids

This example is created by Jeroen van Hunen of the university of Utrecht, faculty of Earth Sciences.

To get this example locally use the command:

```
sepgetex slippingfault
```

To run the example use:

```
sepmesh slippingfault.msh
sepcomp slippingfault.prb
seppost slippingfault.pst
sepview sepplot.001
```

In this example we consider Stokes flow in the earth crust. In this example we have a horizontal fault, which requires the special boundary condition defined by type 914.
In the transition from brittle to ductile deformation, a region exists, where relative displacement is only partly realized by viscous shearing, i.e. internal deformation of the material. Another part of the displacement is concentrated over faults, where two materials slip along each other. In this example, we consider a slipping fault, surrounded by viscous material. Figure 7.1.15.1 shows the curves that define the mesh. The aspect ratio of the box is 3. The fault is defined as a cut in the

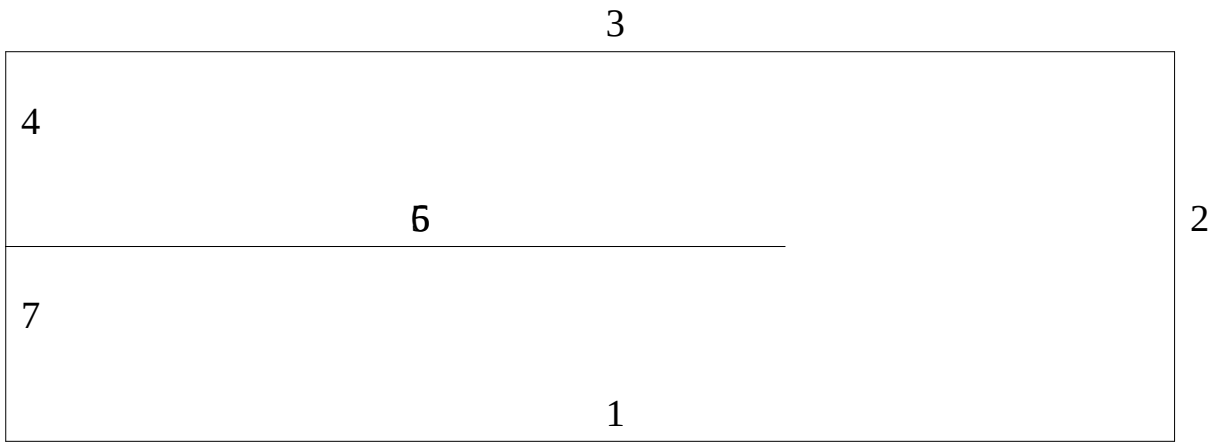


Figure 7.1.15.1: curve numbers in mesh

mesh from $x = 0$ to $x = 2$. Boundaries 5 and 6 define the upper and lower boundary of the fault, respectively.

The boundary conditions are:

- horizontal flows: $v = 0$ at C1 and $v = 1$ at C3
- pressure level uniform at in- and outflow boundaries C2, C4 and C7: $\sigma_n = -p = 0$
- On the fault, we use a discontinuous boundary condition, defined as a special type of mixed boundary condition (see 7.1, type 5):

$$\sigma_t = T_t - C_t(v_t^{upper} - v_t^{lower})$$

We take $T_t = 0$. The boundary condition relates the shear stress σ_t and the velocity jump over the fault $\Delta v = v_t^{upper} - v_t^{lower}$ linearly, using the relation coefficient C_t. The 'upper' and 'lower' boundary are relatively defined, referring to the both sides of the fault: C5 and C6, or C6 and C5, respectively.

The Stokes equation is solved using the penalty function method and extended quadratic elements. In order to apply the discontinuous boundary condition over the fault, quadratic meshconnect elements are defined to connect both sides of the fault. For the internal elements, type number 900 is used, while on the mesh-connect elements, internal elements of number 914 are used. The internal element of type number 914 requires the same coefficients as the boundary element of type 910, which are described in 7.1. Due to the discontinuous boundary condition, the stiffness matrix becomes non-symmetrical.

The following input for sepmesh, sepcomp and seppost is used:

```
#  slippingfault.msh
#
#  mesh file for slipping fault in between two viscous fluids
#  See Manual Standard Elements Section 7.1.15
#
#  To run this file use:
#     sepmesh slippingfault.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      x_left  = 0          # value of x at the left of the box
      x_right = 3          # value of x at the right of the box
      x_fault = 2          # value of x at the end of the fault
      y_low   = 0          # value of y at the bottom of the box
      y_top   = 1          # value of y at the top of the box
      y_fault = 0.5        # value of y at the fault
      c_all   = 1          # relative coarseness in the region
      c_fault_begin = 0.5  # relative coarseness at the start of the fault
      c_fault_end = 0.25   # relative coarseness at the end of the fault
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
   coarse(UNIT=0.2) # coarseness with unit length
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1 = (  x_left ,  y_low  ,  c_all )           # lower-left point
      p2 = (  x_right,  y_low  ,  c_all )           # lower-right point
      p3 = (  x_right,  y_top  ,  c_all )           # upper-right point
      p4 = (  x_left ,  y_top  ,  c_all )           # upper-left point
      p5 = (  x_left ,  y_fault,  c_fault_begin )   # upper-left point of fault
      p6 = (  x_fault,  y_fault,  c_fault_end )     # right point of fault
      p7 = (  x_left ,  y_fault,  c_fault_begin )   # lower-left point of fault
```

```
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=cline2(p1,p2)      # lower boundary
      c2=cline2(p2,p3)      # right-hand boundary
      c3=cline2(p3,p4)      # upper boundary
      c4=cline2(p4,p5)      # upper part left-hand boundary
      c5=cline2(p5,p6)      # upper part fault
      c6=cline2(p6,p7)      # lower part fault
      c7=cline2(p7,p1)      # lower part left-hand boundary

#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
                    # Quadratic triangles are used
      s1=general4(c1,c2,c3,c4,c5,c6,c7)
#
#  Connect elements to element groups
#

   meshsurf
      selm1=(s1)       # all elements in s1 belong to group 1

   meshconnect         # connection elements (group 2)
      celm2=curves2(c5,-c6)        # the elements op and below the fault are
                                   # connected by connection elements
                                   # This is necessary for the special boundary
                                   # condition

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2
end
```

Figure 7.1.15.2 shows the mesh created by sepmesh.

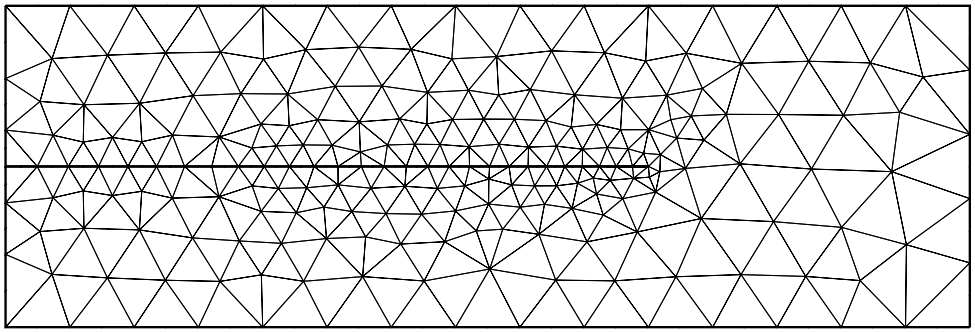


Figure 7.1.15.2: Mesh

```
# slippingfault.prb
#
#  problem file slipping fault in between two viscous fluids
#  penalty method
#  problem is linear
#  See Manual Standard Elements Section 7.1.15
#
#  To run this file use:
#     sepcomp slippingfault.prb
#
#  Reads the file meshoutput
#  Creates the files sepcomp.inf and sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eta   = 1                     # viscosity
      rho   = 1                     # density
      eps   = 1d-6                  # penalty parameter
   vector_names
      velocity
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=(type=900)         # Type number for Navier-Stokes, without swirl
                                # penalty function approach
                                # See Standard problems Section 7.1
      elgrp2=(type=914)         # Type number for discontinuous boundary
                                # condition
                                # See Standard problems Section 7.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
            degfd2=curves100(c7)     # inflow (lower part), skip start point
                                     # vertical component given
            degfd2=curves200(c4)     # inflow (upper part), skip end point
                                     # vertical component given
     degfd1,degfd2=curves (c1)       # bottom, full elcoity given
     degfd1,degfd2=curves (c3)       # top, full elcoity given
            degfd2=curves (c5)       # fault upper part, normal component given
            degfd2=curves (c6)       # fault lower part, normal component given
            degfd2=curves (c2)       # outflow
                                     # vertical component given
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4
```

```
matrix
                        # non-symmetrical profile matrix (default)
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  Since this is the default one, it may be skipped
#
structure                   # See Users Manual Section 3.2.3
   prescribe_boundary_conditions, velocity
   solve_linear_system, velocity
   output
end

# Put the essential boundary conditions into the velocity vector
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions, sequence_number=1
     curves (c1), degfd1=(value=0)      # u = 0 at bottom
     curves (c3), degfd1=(value=1)      # u = q at top
end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )   # The coefficients are defined by 20 parameters
      icoef2 = 1         # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0         # 5:  Type of linearization (0=Stokes flow)
      coef6  =  eps      # 6:  Penalty function parameter eps
      coef7  =  rho      # 7:  Density
      coef12 =  eta      #12:  Value of eta  (viscosity)
   elgrp2 ( nparm=15 )   # For the boundary condition 15 parameters are needed
      coef9 = 1          # 9:  c_x is defined as a constant
end

# input for linear solver
# See Users Manual Section 3.2.8

solve
   direct_solver = profile
end

end_of_sepran_input

#  slippingfault.pst
#  Input file for postprocessing for
#  slipping fault in between two viscous fluids
#  See Manual Standard Elements Section 7.1.15
#
#
#  To run this file use:
```

```
#     seppost slippingfault.pst > slippingfault.out
#
#  Reads the files meshoutput and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2

   define plot parameters = norotate
   plot contour velocity degfd=1, noplot_legenda, noplot_scales
   plot intersection velocity degfd=1, origin=(0.5,0.0) angle=90 //
        textx='y', texty='velocity', length=7, noaxis
end
```

Contour plots of the velocity field and vertical cross section at $x = 0.5$ of the horizontal velocity field in figures 7.1.15.3 to 7.1.15.5.

7.1.16 Application of some 2D and 3D elements to a simple Couette flow

In this section we consider a very simple Couette flow (Cartesian co-ordinates) for low Reynolds numbers. The exact solution is a linear velocity profile perpendicular to the flow direction and a zero pressure field. The reason to solve this simple problem is that it is an easy test on correctness of elements. In the next Section (7.1.17), it is shown how this example can be extended with friction. In order to get these examples into your local directory use the command

```
sepgetex couettexx
```

where xx is a 2 digit number. The following numbers are available:

number	shape	type	description
11	4	900	extended quadratic triangle, penalty method
12	5	900	linear quadrilateral, penalty method
13	6	900	biquadratic quadrilateral, penalty method
21	6	902	biquadratic quadrilateral, integrated method
22	7	902	extended quadratic triangle, integrated method
23	9	902	bilinear quadrilateral, integrated method
31	7	901	extended quadratic triangle, integrated method (elimination)
41	3	903	linear triangle, Taylor Hood
42	4	903	quadratic triangle, Taylor Hood
43	6	903	biquadratic quadrilateral, Taylor Hood
44	10	903	extended linear triangle, Taylor Hood
51	14	900	extended triquadratic hexahedron, penalty method
81	11	903	linear tetrahedron, Taylor Hood

To run this example use:

```
sepmesh couettexx.msh
view mesh
sepcomp < couettexx.prb
seppost couettexx.pst
view results
```

Mark that the possibilities 5x to 8x are three-dimensional. In this case the flow is linear in the z-direction and constant in the y-direction.

Figure 7.1.8.1 shows the channel and the corresponding curves.
The tangential velocity at the inlet and outlet are equal to zero, the normal velocity is not prescribed. The normal stress at inlet and outlet is made equal to 0, so no extra information for the normal components is necessary.
At the lower wall we have a zero velocity and at the upper wall the tangential velocity is 1 and the normal velocity 0.
In all our examples we use a 8×8 linear or 8×8 quadratic subdivision in elements.

The exact solution is a zero v-velocity and a linearly varying u-velocity: $u(x, y) = y$. The corresponding pressure is equal to 0. We consider the only give the input for the "11" example, all other ones are very similar. See also the channel problem in Section 7.1.8.

shape = 4 // The input for program SEPMESH is given in the following input file (couette11.msh):

```
#  couette11.msh
#
#  mesh file for 2d couette problem
#  See Manual Standard Elements Section 7.1.16
```

```
#
#  To run this file use:
#     sepmesh couette11.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1        # width of the channel
      length = 4       # length of the channel
   integers
      n = 4            # number of elements in length direction
      m = 4            # number of elements in width direction
      lin = 2          # quadratic elements
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)              # Left under point
      p2=( length,0)        # Right under point
      p3=( length, width)   # Right upper point
      p4=(0, width)         # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line  lin (p1,p2,nelm= n)     # lower wall
      c2=line  lin (p2,p3,nelm= m)     # outflow boundary
      c3=line  lin (p3,p4,nelm= n)     # upper wall
      c4=line  lin (p4,p1,nelm= m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
                    # Quadratic triangles are used
      s1=rectangle4(c1,c2,c3,c4)

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2

end
```

The input file for SEPCOMP is given by the file couette11.prb:

```
# couette11.prb
#
#  problem file for 2d couette problem
#  penalty function approach
```

```
#  problem is stationary and non-linear
#  See Manual Standard Elements Section 7.1.16
#
#  To run this file use:
#     sepcomp couette11.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1d-6              # penalty parameter for Navier-Stokes
      rho       = 1                 # density
      eta       = 0.01              # viscosity
      v_top     = 1                 # velocity on top_wall
   vector_names
      velocity
      pressure
      stress
   variables
      pressure_int
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=900               # Type number for Navier-Stokes, without swirl
                               # See Standard problems Section 7.1
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      degfd1,degfd2=curves(c1) # Fixed under wall
      degfd1,degfd2=curves(c3) # Fixed upper wall
      degfd2       =curves(c4) # inflow (v-component given)
      degfd2       =curves(c2) # Outstream boundary (v-component given)
                               # All not prescribed boundary conditions
                               # satisfy corresponding stress is zero
end
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because the integral of the pressure over the boundary
#  is required
#
structure                  # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions, velocity
   solve_nonlinear_system, velocity
 # Compute the pressure
```

```
   derivatives, seq_deriv=1, pressure
 # Compute the stress
   derivatives, seq_deriv=2, stress
 # Compute the integral of the pressure over curve c2 (outflow boundary)
   boundary_integral, pressure, pressure_int
   print pressure_int, text = 'integral of pressure over curve c2'
 # Write the results to a file
   output
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c3), degfd1, value= v_top   # The u-component of the velocity at
                                      # the top wall is 1
                                      # The rest of the vector is 0

end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef6  =  eps        # 6:  Penalty function parameter eps
      coef7  =  rho        # 7:  Density
      coef12 =  eta        #12:  Value of eta (viscosity)
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1//
   at_error return
   equation 1
```

```
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end
#
#  Define information with respect to the boundary integral to be computed
#  See Users Manual, Section 3.2.14
#
boundary_integral, sequence_number = 1
   ichint = 1                 # Standard integration
   curves = c4                # integral over curve c4
end

# compute pressure and stress
# See Users Manual, Section 3.2.11

derivatives, sequence_number = 1
   icheld=7          # icheld=7, pressure in nodes
                     # See Standard problems Section 7.1
end
derivatives, sequence_number = 2
   icheld=6          # icheld=6, stress in nodes
                     # See Standard problems Section 7.1
end
end_of_sepran_input
```

The standard nonlinear algorithm, i.e. start with Stokes, do one step Picard and finally use Newton is applied. However, for this particular problem the solution is reached in two steps due to the fact that the convective terms do not play a role.
The solution with this element is of course exact up to an accuracy of the order of 10^{-6}, which is the penalty function parameter.
The postprocessing input file couette11.pst, which produces the pictures shown before is defined by:

```
#  couette11.pst
#  Input file for postprocessing for couette problem
#  See Manual Standard Elements Section 7.1.16
#
#
#  To run this file use:
#     seppost couette11.pst > couette11.out
#
#  Reads the files meshoutput and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2

#
#  compute the stream function
#  See Users Manual Section 5.2
#  store in stream_function

   compute stream_function = stream function velocity

#  Plot the results
#  See Users Manual Section 5.4
```

```
   plot vector velocity              # Vector plot of velocity
   plot contour pressure             # Contour plot of pressure
   plot coloured contour pressure
   plot contour stream_function      # Contour plot of stream function
   plot coloured contour stream_function

#  Print the results
#  See Users Manual Section 5.3

   print vector stress              # Print of stress

end
```

7.1.17 Application of some 2D and 3D elements to a simple Couette flow with friction

In this section we consider also very simple Couette flow (Cartesian co-ordinates) for low Reynolds numbers. The exact solution is a linear velocity profile perpendicular to the flow direction and a zero pressure field. The difference with the example in Section (7.1.16), is that in this case the velocity of upper and under surface are not prescribed to flow by means of a no-slip condition, but that a friction boundary condition is used.
In one example (91) the friction coefficient at the bottom is made so large that in fact a no-slip condition is simulated. In this example an iterative linear solver is used and to get a good convergence the matrix must be scaled.
In order to get these examples into your local directory use the command

```
sepgetex couettefrictxx
```

where xx is a 2 digit number. The following numbers are available:

number	shape	type	description
11	4	900	extended quadratic triangle, penalty method
12	5	900	linear quadrilateral, penalty method
13	6	900	biquadratic quadrilateral, penalty method
21	6	902	biquadratic quadrilateral, integrated method
22	7	902	extended quadratic triangle, integrated method
23	9	902	bilinear quadrilateral, integrated method
31	7	901	extended quadratic triangle, integrated method (elimination)
41	3	903	linear triangle, Taylor Hood
42	4	903	quadratic triangle, Taylor Hood
43	6	903	biquadratic quadrilateral, Taylor Hood
44	10	903	extended linear triangle, Taylor Hood
51	14	900	extended triquadratic hexahedron, penalty method
81	11	903	linear tetrahedron, Taylor Hood
91	11	903	linear tetrahedron, Taylor Hood, no-slip at bottom

To run this example use:

```
sepmesh couettefrictxx.msh
view mesh
seplink couettefrictxx
couettefrictxx < couettefrictxx.prb
seppost couettefrictxx.pst
view results
```

Mark that the possibilities 5x to 9x are three-dimensional. In this case the flow is linear in the z-direction and constant in the y-direction.

Figure 7.1.8.1 shows the channel and the corresponding curves.
The tangential velocity at the inlet and outlet are equal to zero, the normal velocity is not prescribed. The normal stress at inlet and outlet is made equal to 0, so no extra information for the normal components is necessary.
The lower wall has a zero velocity and at the upper has velocity 1.
The friction is modeled by the boundary condition $c_t \mathbf{v}_t + \sigma_t = \mathbf{f} \cdot \mathbf{t}$, with $\mathbf{t}$ the tangential vector, c_t the friction coefficient and $\mathbf{t}$ the velocity of the surface.
This means that in R^2 we use the natural boundary condition with ILOAD=0 and in R^3 with ILOAD=3.
In all our examples we use a 8×8 linear or 8×8 quadratic subdivision in elements.

The exact solution is a zero v-velocity and a linearly varying u-velocity: $u(x, y) = 0.1 + 0.8y$. The

corresponding pressure is equal to 0.
The friction coefficient has value 8, in order to get this exact solution. We consider the only give the input for the "11" and "91" example, all other ones are very similar. See also the channel problem in Section 7.1.8.

Example 11: Pure friction The input for program SEPMESH is given in the following input file (couettefrict11.msh):

```
#  couettefrict11.msh
#
#  mesh file for 2d couette problem with friction
#  See Manual Standard Elements Section 7.1.17
#
#  To run this file use:
#     sepmesh couettefrict11.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1        # width of the channel
      length = 4       # length of the channel
   integers
      n = 4            # number of elements in length direction
      m = 4            # number of elements in width direction
      lin = 2          # quadratic elements
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)              # Left under point
      p2=( length,0)        # Right under point
      p3=( length, width)   # Right upper point
      p4=(0, width)         # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line  lin (p1,p2,nelm= n)     # lower wall
      c2=line  lin (p2,p3,nelm= m)     # outflow boundary
      c3=line  lin (p3,p4,nelm= n)     # upper wall
      c4=line  lin (p4,p1,nelm= m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
                    # Quadratic triangles are used
      s1=rectangle4(c1,c2,c3,c4)
```

```
   plot                          # make a plot of the mesh
                                 # See Users Manual Section 2.2

end
```

The input file for SEPCOMP is given by the file couettefrict11.prb:

```
# couettefrict11.prb
#
#  problem file for 2d couette problem with friction
#  penalty function approach
#  problem is stationary and non-linear
#  See Manual Standard Elements Section 7.1.17
#
#  To run this file use:
#     sepcomp couettefrict11.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1d-6              # penalty parameter for Navier-Stokes
      rho       = 1                 # density
      eta       = 0.01              # viscosity
      mu        = 8                 # Friction coefficient
      v_bot_tang = 0                # Tangential velocity of lower wall
      v_top_tang = -1               # Tangential velocity of upper wall
                                    # Mark that the velocity is negative
                                    # because the tangential vector = (-1,0)
                                    # and the flow is to the right
      c_upp = mu*eta                # Friction coefficient at upper wall
                                    # mu eta
      c_low = c_upp                 # Friction coefficient at lower wall
                                    # mu eta
      sigma_upp = c_upp*v_top_tang  # Friction coefficient times velocity
                                    # at upper wall
      sigma_low = c_upp*v_bot_tang  # Friction coefficient time velocity
                                    # at lower wall
   vector_names
      velocity
      pressure
      stress
   variables
      pressure_int
end
#
#  Define the type of problem to be solved
#
problem                  # See Users Manual Section 3.2.2
```

```
   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=900                # Type number for Navier-Stokes, without swirl
                                # See Standard problems Section 7.1
   natbouncond                  # Define the type numbers for the natural
                                # boundary conditions, i.e. the boundary
                                # conditions: c u_t + sigma_t = given
      bngrp1 = 910              # Type number for Natural boundary condition
      bngrp2 = 910
   bounelements                 # Define where the natural boundary conditions
                                # must be applied
      belm1 = curves(c1)        # lower wall
      belm2 = curves(c3)        # upper wall
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      degfd1=curves(c1)         # Lower wall: u_n = 0
      degfd1=curves(c3)         # Upper wall: u_n = 0
      degfd2=curves 300 (c4)    # inflow (v-component given)
                                # The initial and end point are excluded to
                                # avoid a conflict with the local transformations
      degfd2=curves 300 (c2)    # Outstream boundary (v-component given)
                                # All not prescribed boundary conditions
                                # satisfy corresponding stress is zero
    local_transform             # Local transformations to get the normal
                                # and tangential components as first resp. second
                                # unknown at the walls
       curves c1                # standard transformation at lower wall
       curves c3                # standard transformation at upper wall
end
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because the integral of the pressure over the boundary
#  is required
#
structure                    # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions, velocity
   solve_nonlinear_system, velocity
 # Compute the pressure
   derivatives, seq_deriv=1, pressure
 # Compute the stress
   derivatives, seq_deriv=2, stress
 # Compute the integral of the pressure over curve c2 (outflow boundary)
   boundary_integral, pressure, pressure_int
   print pressure_int, text = 'integral of pressure over curve c2'
 # Write the results to a file
   output
end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
```

```
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef6  =  eps        # 6:  Penalty function parameter eps
      coef7  =  rho        # 7:  Density
      coef12 =  eta        #12:  Value of eta (viscosity)
   bngrp1 ( nparm=15 )     # The coefficients are defined by 15 parameters
                           # boundary elements at lower wall
      coef7 =  sigma_low   # friction times wall velocity
      coef10 =  c_low      # friction coefficient
   bngrp2 ( nparm=15 )     # The coefficients are defined by 15 parameters
                           # boundary elements at upper wall
      coef7 =  sigma_upp   # friction times wall velocity
      coef10 =  c_upp      # friction coefficient
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1//
   at_error return
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end
#
#  Define information with respect to the boundary integral to be computed
#  See Users Manual, Section 3.2.14
#
boundary_integral, sequence_number = 1
   ichint = 1                 # Standard integration
   curves = c4                # integral over curve c4
end

# compute pressure and stress
# See Users Manual, Section 3.2.11

derivatives, sequence_number = 1
   icheld=7          # icheld=7, pressure in nodes
```

```
                    # See Standard problems Section 7.1
end
derivatives, sequence_number = 2
   icheld=6         # icheld=6, stress in nodes
                    # See Standard problems Section 7.1
end

end_of_sepran_input
```

The standard nonlinear algorithm, i.e. start with Stokes, do one step Picard and finally use Newton is applied. However, for this particular problem the solution is reached in two steps due to the fact that the convective terms do not play a role.
The solution with this element is of course exact up to an accuracy of the order of 10^{-6}, which is the penalty function parameter.
The postprocessing input file couettefrict11.pst, which produces the pictures shown before is defined by:

```
#  couettefrict11.pst
#  Input file for postprocessing for couette problem with friction
#  See Manual Standard Elements Section 7.1.17
#
#
#  To run this file use:
#     seppost couettefrict11.pst > couettefrict11.out
#
#  Reads the files meshoutput and sepcomp.out
#
postprocessing                   # See Users Manual Section 5.2

#
#  compute the stream function
#  See Users Manual Section 5.2
#  store in stream_function

   compute stream_function = stream function velocity

#  Plot the results
#  See Users Manual Section 5.4

   plot vector velocity             # Vector plot of velocity
   plot contour pressure            # Contour plot of pressure
   plot coloured contour pressure
   plot contour stream_function     # Contour plot of stream function
   plot coloured contour stream_function

#  Print the results
#  See Users Manual Section 5.3

   print vector stress             # Print of stress

end
```

Example 91: Friction at upper face and almost slip at lower face This is the 3D equivalent of the channel given above.
At the bottom we have a very large friction coefficient. Effectively this results in a (almost)

no-slip condition.
The input for program SEPMESH is given in the following input file (couettefrict91.msh):

```
#  couettefrict11.msh
#
#  mesh file for 2d couette problem with friction
#  See Manual Standard Elements Section 7.1.17
#
#  To run this file use:
#     sepmesh couettefrict11.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      width = 1        # width of the channel
      length = 4       # length of the channel
   integers
      n = 4            # number of elements in length direction
      m = 4            # number of elements in width direction
      lin = 2          # quadratic elements
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=(0,0)              # Left under point
      p2=( length,0)        # Right under point
      p3=( length, width)   # Right upper point
      p4=(0, width)         # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line  lin (p1,p2,nelm= n)     # lower wall
      c2=line  lin (p2,p3,nelm= m)     # outflow boundary
      c3=line  lin (p3,p4,nelm= n)     # upper wall
      c4=line  lin (p4,p1,nelm= m)     # inflow boundary
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
                    # Quadratic triangles are used
      s1=rectangle4(c1,c2,c3,c4)

   plot                          # make a plot of the mesh
                                 # See Users Manual Section 2.2
```

```
end
```

The problem is solved by an iterative linear solver. Due to the large friction coefficient at the bottom, we have large elements in the corresponding rows of the matrix and right-hand side. If we apply the iterative solver in that case, the residual starts with a large value and after one step is much smaller. If the standard accuracy and termination criterion is used, this means that the iteration is stopped after one step, although the solution is far from accurate. In order to avoid that problem, the matrix is scaled by a row scaling. The result is the absence of large elements in matrix and right-hand side and a smooth convergence.
Since actually the non-linear iteration should be finished after two iterations, we have increased the accuracy of the linear solver to 1d-5.
The input file for sepcomp is given below:

```
# couettefrict11.prb
#
#  problem file for 2d couette problem with friction
#  penalty function approach
#  problem is stationary and non-linear
#  See Manual Standard Elements Section 7.1.17
#
#  To run this file use:
#     sepcomp couettefrict11.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps       = 1d-6              # penalty parameter for Navier-Stokes
      rho       = 1                 # density
      eta       = 0.01              # viscosity
      mu        = 8                 # Friction coefficient
      v_bot_tang = 0                # Tangential velocity of lower wall
      v_top_tang = -1               # Tangential velocity of upper wall
                                    # Mark that the velocity is negative
                                    # because the tangential vector = (-1,0)
                                    # and the flow is to the right
      c_upp                         # Friction coefficient at upper wall
                                    # mu eta
      c_low                         # Friction coefficient at lower wall
                                    # mu eta
      sigma_upp                     # Friction coefficient times velocity
                                    # at upper wall
      sigma_low                     # Friction coefficient time velocity
                                    # at lower wall
   vector_names
      velocity
      pressure
      stress
   variables
      pressure_int
end
```

```
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=900               # Type number for Navier-Stokes, without swirl
                               # See Standard problems Section 7.1
   natbouncond                 # Define the type numbers for the natural
                               # boundary conditions, i.e. the boundary
                               # conditions: c u_t + sigma_t = given
      bngrp1 = 910             # Type number for Natural boundary condition
      bngrp2 = 910
   bounelements                # Define where the natural boundary conditions
                               # must be applied
      belm1 = curves(c1)       # lower wall
      belm2 = curves(c3)       # upper wall
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      degfd1=curves(c1)        # Lower wall: u_n = 0
      degfd1=curves(c3)        # Upper wall: u_n = 0
      degfd2=curves 300 (c4)   # inflow (v-component given)
                               # The initial and end point are excluded to
                               # avoid a conflict with the local transformations
      degfd2=curves 300 (c2)   # Outstream boundary (v-component given)
                               # All not prescribed boundary conditions
                               # satisfy corresponding stress is zero
    local_transform            # Local transformations to get the normal
                               # and tangential components as first resp. second
                               # unknown at the walls
       curves c1               # standard transformation at lower wall
       curves c3               # standard transformation at upper wall
end
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because the integral of the pressure over the boundary
#  is required
#
structure                  # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions, velocity
   solve_nonlinear_system, velocity
 # Compute the pressure
   derivatives, seq_deriv=1, pressure
 # Compute the stress
   derivatives, seq_deriv=2, stress
 # Compute the integral of the pressure over curve c2 (outflow boundary)
   boundary_integral, pressure, pressure_int
   print pressure_int, text = 'integral of pressure over curve c2'
 # Write the results to a file
   output
end
```

```
#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef6  =  eps        # 6:  Penalty function parameter eps
      coef7  =  rho        # 7:  Density
      coef12 =  eta        #12:  Value of eta (viscosity)
   bngrp1 ( nparm=15 )     # The coefficients are defined by 15 parameters
                           # boundary elements at lower wall
      coef7 =  sigma_low   # friction times wall velocity
      coef10 =  c_low      # friction coefficient
   bngrp2 ( nparm=15 )     # The coefficients are defined by 15 parameters
                           # boundary elements at upper wall
      coef7 =  sigma_upp   # friction times wall velocity
      coef10 =  c_upp      # friction coefficient
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1            # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2            # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1//
   at_error return
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end
#
#  Define information with respect to the boundary integral to be computed
#  See Users Manual, Section 3.2.14
#
boundary_integral, sequence_number = 1
   ichint = 1                # Standard integration
   curves = c4               # integral over curve c4
end
```

```
# compute pressure and stress
# See Users Manual, Section 3.2.11

derivatives, sequence_number = 1
   icheld=7          # icheld=7, pressure in nodes
                     # See Standard problems Section 7.1
end
derivatives, sequence_number = 2
   icheld=6          # icheld=6, stress in nodes
                     # See Standard problems Section 7.1
end

end_of_sepran_input
```

And the input file for seppost:

```
#  couettefrict11.pst
#  Input file for postprocessing for couette problem with friction
#  See Manual Standard Elements Section 7.1.17
#
#
#  To run this file use:
#     seppost couettefrict11.pst > couettefrict11.out
#
#  Reads the files meshoutput and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2

#
#  compute the stream function
#  See Users Manual Section 5.2
#  store in stream_function

   compute stream_function = stream function velocity

#  Plot the results
#  See Users Manual Section 5.4

   plot vector velocity              # Vector plot of velocity
   plot contour pressure             # Contour plot of pressure
   plot coloured contour pressure
   plot contour stream_function      # Contour plot of stream function
   plot coloured contour stream_function

#  Print the results
#  See Users Manual Section 5.3

   print vector stress              # Print of stress

end
```

7.1.18 Some examples of how to apply pressure-correction

In this section we show the use of the pressure-correction method to solve the time-dependent Navier-Stokes equations.
The following examples are available

channelintsthpc42 (7.1.18.1) Solves the channel flow using the Navier-Stokes equations and pressure-correction. A direct symmetric linear solver is used, and quadratic Taylor-Hood elements.

backwrd2_thpc (7.1.18.2) Solves the 2d backward facing step using pressure correction and an iterative solver for the linear equations

7.1.18.1 Channel flow, solved by Navier-Stokes, direct solver

In order to get this example into your local directory use the command

```
sepgetex channelinsthpc42
```

To run this example use:

```
sepmesh channelinsthpc42.msh
view mesh
sepcomp channelinsthpc42.prb
seppost channelinsthpc42.pst
view results
```

The example is a simple 2d channel flow, with velocity from left to right. The upper and lower boundaries are fixed walls and at the outlet we assume parallel flow. The velocity at the inlet is prescribed by a quadratic velocity profile. The problem is time-dependent. At $t = 0$ we set the velocity $\mathbf{u}$ equal to zero, except for the inflow boundary and the pressure p also to zero. After the computations, the solution must have been converged to the stationary solution. Since the initial condition is not divergence-free, we may expect a transient and this is exactly what happens. In the special case of a channel flow, the convective terms of the stationary solution vanish due to the simple quadratic velocity field in x-direction and zero velocity in y-direction. For that reason we may solve this problem without convection and all matrices will be symmetrical positive definite. However, running Navier-Stokes is only a small change in the input, as you can see below the input for sepcomp.

The input for program SEPMESH is given in the following input file (channelinstpc42.msh):

```
#  channelinsthpc42.msh
#
#
#  Instationary flow in channel (2D case)
#  Taylor-Hood elements are used
#  pressure correction method
#
#  See Manual Standard Elements Section 7.1.10
#  See Manual Examples Section 7.1.18
#
#  Quadratic triangles
#
#  To create the mesh run:
#
#  sepmesh channelinsthpc42.msh
#
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants
    reals
       length    = 4         # length of channel
       height    = 1         # Height of channel
                             # centre of the cylinder
    integers
       lin       = 2         # Quadratic line elements
       sur       = 4         # Quadratic triangles
       n_hor     = 4         # number of elements along the outer boundary
                             # horizontal direction
       n_ver     = 4         # number of elements along the outer boundary
                             # vertical direction
       inflow    = 4         # curve number of inflow boundary
       outlet    = 2         # curve number of outlet boundary
       wall      = 5         # curve number of wall
end
#
#  Define the mesh
#
mesh2d            # See Users Manual Section 2.2
#
#  user points
#
points             # See Users Manual Section 2.2
      p1 = (0,0)                # Left-under point
      p2 = (length,0)          # Right-under point
      p3 = (length,height)    # Right-upper point
      p4 = (0,height)          # Left-upper point
#
#   curves
#
   curves          # See Users Manual Section 2.3
```

```
      c1=line lin (p1,p2,nelm=n_hor)         # Lower boundary
      c outlet=line lin (p2,p3,nelm=n_ver)   # Outflow boundary
      c3=line lin (p3,p4,nelm=n_hor)         # Upper boundary
      c inflow=line lin (p4,p1,nelm=n_ver)   # Inflow boundary
      c wall = curves (c1, c3)               # wall
 #
 #  surface
 #
   surfaces       # See Users Manual Section 2.4
      s1=rectangle sur (c1,c2,c3,c4)       # outer region
   plot           # make a plot of the mesh
                  # See Users Manual Section 2.2

end
```

The input file for SEPCOMP is given by the file channelinsthpc42.prb:

```
#  channelinsthpc42.prb
#
#
#  Instationary flow in channel (2D case)
#  Taylor-Hood elements are used
#  pressure correction method
#
#  See Manual Standard Elements Section 7.1.10
#  See Manual Examples Section 7.1.18
#
#  Quadratic triangles
#
#  To run this file use:
#     sepcomp channelinsthpc42.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      inflow = 4                    # curve number of inflow boundary
      outlet = 2                    # curve number of outlet boundary
      wall   = 5                    # curve number of wall
   reals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
      t0        = 0                 # initial time
      dt        = 0.1               # time step
      t1        = 5                 # end time
      tout0     = t0                # First time that a result is written
      toutend   = t1                # End time for writing
      toutstep  =  5*dt             # In each 5^th time step the result is written
      umax      = 1                 # maximum velocity at inflow
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#  In this case we have 2 problems, 1 for the velocity
#  and one for the pressure.
#  Both are solved subsequently
#
problem 1                 # See Users Manual Section 3.2.2
                          # solves the velocity (momentum equations: predictor)

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=905               # Type number for Navier-Stokes, without swirl
```

```
                              # pressure correction method
                              # Taylor-Hood elements
                              # See Standard problems Section 7.1
   essbouncond                # Define where essential boundary conditions are
                              # given (not the value)
                              # See Users Manual Section 3.2.2
                              # Only velocities are prescribed, not the
                              # pressures
      degfd1,degfd2=curves(c wall) # Fixed wall
      degfd1,degfd2=curves(c inflow) # inflow
      degfd2       =curves(c outlet) # Outstream boundary (v-component given)
                              # All not prescribed boundary conditions
                              # satisfy corresponding stress is zero

problem 2                 # See Users Manual Section 3.2.2
                              # solves the pressure equation

   types                      # Define types of elements,
                              # See Users Manual Section 3.2.2
      elgrp1=906              # Type number for pressure equation used in
                              # case of Navier-Stokes, pressure correction
                              # Taylor Hood approach
                              # See Standard problems Section 7.1
   essbouncond                # Define where essential boundary conditions are
                              # given (not the value)
                              # See Users Manual Section 3.2.2
                              # The pressure is prescribed at the outlet
                              # In other boundaries we have dp/dn = 0
       curves(c outlet)       # Outlet boundary (p=0)
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4
matrix
   symmetric, problem = 1     # Symmetrical profile matrix for velocity
   symmetric, problem = 2     # symmetrical profile matrix for pressure
                              # So a direct method will be applied
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions  # velocity only

   curves(c inflow), degfd1, quadratic, max = umax   # The u-component of the
                                  #  velocity at instream is quadratic
                                  # The rest of the vector is 0

end

#  Create pressure vector and set equal to 0

create vector, problem = 2
end
```

```
#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  0: Stokes,  1: Picard
      icoef6 = pressure    # 6:  iseqpress
      coef7  = rho         # 7:  Density
      coef12 = eta         #12:  Value of eta (viscosity)
end

#  Input for definition of pressure correction
#  In this case only defaults are used
#  See Users Manual Section 3.2.15

pressure_correction
end

#  Definition of time integration
#  See Users Manual Section 3.2.22

time_integration
   method = euler_implicit         # Integration by the Euler implicit method
   tinit = t0                      # Initial time
   tend = t1                       # End time
   tstep = dt                      # Time step
   toutinit = tout0                # First time that a result is written
   toutend = toutend               # End time for writing
   toutstep = toutstep             # time steps for writing
   boundary_conditions = constant  # The boundary conditions do not depend on
                                   # time
   seq_boundary_conditions = 1     # Sequence number for the input of the
                                   # essential boundary conditions
   mass_matrix = constant          # Time-independent mass matrix

end

# Description of how the Navier-Stokes equations are solved
# See Users Manual, Section 3.2.22

navier_stokes
   method = pressure_correction       # solve by pressure correction
      seq_pressure_correction = 1     # sequence number of pressure correction
                                      # input
end

#  Input for the linear solver
#  See Users Manual Section 3.2.8

solve
   positive_definite
end
```

```
#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#

structure                  # See Users Manual Section 3.2.3

  # Compute start vector for the flow by filling boundary conditions
   prescribe_boundary_conditions, sequence_number=1, vector=velocity
   create_vector, sequence_number=1, vector=pressure

  # Time loop
   start_time_loop

   # One time step to compute the velocity and the pressure,
   # using pressure correction

     navier_stokes

     output

   end_time_loop

end

end_of_sepran_input
```

In this case also convection substepping is allowed.
To get the corresponding files into your directory use:

```
sepgetex channelinstcspc42
```

The corresponding problem file reads

```
#  channelinstcspc42.prb
#
#
#  Instationary flow in channel (2D case)
#  Taylor-Hood elements are used
#  pressure correction method and convection substepping
#
#  See Manual Standard Elements Section 7.1.10
#  See Manual Examples Section 7.1.18
#
#  Quadratic triangles
#
#  To run this file use:
#     sepcomp channelinstcspc42.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      inflow = 4                    # curve number of inflow boundary
      outlet = 2                    # curve number of outlet boundary
      wall   = 5                    # curve number of wall
      mstep  = 1                    # Number of convection substeps
   reals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
      t0        = 0                 # initial time
      dt        = 0.1               # time step
      t1        = 5                 # end time
      tout0     = t0                # First time that a result is written
      toutend   = t1                # End time for writing
      toutstep  =  5*dt             # In each 5^th time step the result is written
      umax      = 1                 # maximum velocity at inflow
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#  In this case we have 2 problems, 1 for the velocity
#  and one for the pressure.
#  Both are solved subsequently
#
```

```
problem 1                 # See Users Manual Section 3.2.2
                          # solves the velocity (momentum equations: predictor)

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=905               # Type number for Navier-Stokes, without swirl
                               # pressure correction method
                               # Taylor-Hood elements
                               # See Standard problems Section 7.1
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
                               # Only velocities are prescribed, not the
                               # pressures
      degfd1,degfd2=curves(c wall) # Fixed wall
      degfd1,degfd2=curves(c inflow) # inflow
      degfd2       =curves(c outlet) # Outstream boundary (v-component given)
                               # All not prescribed boundary conditions
                               # satisfy corresponding stress is zero

problem 2                 # See Users Manual Section 3.2.2
                               # solves the pressure equation

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=906               # Type number for pressure equation used in
                               # case of Navier-Stokes, pressure correction
                               # Taylor Hood approach
                               # See Standard problems Section 7.1
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
                               # The pressure is prescribed at the outlet
                               # In other boundaries we have dp/dn = 0
       curves(c outlet)        # Outlet boundary (p=0)
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4
matrix
   symmetric, problem = 1     # Symmetrical profile matrix for velocity
   symmetric, problem = 2     # symmetrical profile matrix for pressure
                              # So a direct method will be applied
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions  # velocity only

   curves(c inflow), degfd1, quadratic, max = umax   # The u-component of the velocity at
                                    # instream is quadratic
                                    # The rest of the vector is 0
```

```
end

#  Create pressure vector and set equal to 0

create vector, problem = 2
end

#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients, sequence_number = 1     #  Stokes
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  0: Stokes,  1: Picard
      icoef6 = pressure    # 6:  iseqpress
      coef7  = rho         # 7:  Density
      coef12 = eta         #12:  Value of eta (viscosity)
end

coefficients, sequence_number = 2, problem = 1     #  Convection step
  # The problem number is required, because it differs from the sequence number
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 2           # 5:  2: Newton, necessary to define u^n grad u^n
      coef7  = rho         # 7:  Density
      coef12 = eta         #12:  Value of eta (viscosity)
end


#  Input for definition of pressure correction
#  In this case only defaults are used
#  See Users Manual Section 3.2.15

pressure_correction
   convection_treatment = substeps      # Use convection substepping
   number_of_substeps = mstep           # number of substeps
   seq_vel_coefficients = 1             # sequence number of velocity
                                        # coefficients input
   seq_conv_coefficients = 2            # sequence number of convection
                                        # coefficients input
   seq_time_integration = 1             # sequence number of time integration
end

#  Definition of time integration
#  See Users Manual Section 3.2.22

time_integration
   method = euler_implicit         # Integration by the Euler implicit method
   tinit = t0                      # Initial time
   tend = t1                       # End time
   tstep = dt                      # Time step
   toutinit = tout0                # First time that a result is written
   toutend = toutend               # End time for writing
   toutstep = toutstep             # time steps for writing
```

```
   boundary_conditions = constant  # The boundary conditions do not depend on
                                   # time
   seq_boundary_conditions = 1     # Sequence number for the input of the
                                   # essential boundary conditions
   mass_matrix = constant          # Time-independent mass matrix

end

# Description of how the Navier-Stokes equations are solved
# See Users Manual, Section 3.2.22

navier_stokes
   method = pressure_correction       # solve by pressure correction
      seq_pressure_correction = 1     # sequence number of pressure correction
                                      # input
end

#  Input for the linear solver
#  See Users Manual Section 3.2.8

solve
   positive_definite
end
#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#

structure                  # See Users Manual Section 3.2.3

  # Compute start vector for the flow by filling boundary conditions
   prescribe_boundary_conditions, sequence_number=1, vector=velocity
   create_vector, sequence_number=1, vector=pressure

  # Time loop
   start_time_loop

   # One time step to compute the velocity and the pressure,
   # using pressure correction

     navier_stokes

     output

   end_time_loop

end

end_of_sepran_input
```

The postprocessing input file channelinsthpc42.pst is defined by:

```
#  channelinsthpc42.pst
#
#  Instationary flow in channel (2D case)
#  Taylor-Hood elements are used
#  pressure correction method
#
#  See Manual Standard Elements Section 7.1.10
#  See Manual Examples Section 7.1.18
#
#  Quadratic triangles
#
#
#  To run this file use:
#     seppost channelinsthpc42.pst > channelinsthpc42.out
#
#  Reads the files meshoutput and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2
#
#
#  compute the stream function
#  See Users Manual Section 5.2
#  store in stream_function

   compute stream_function = stream function velocity

#  Plot the results
#  See Users Manual Section 5.4

   time = (0, 10)
      plot vector velocity              # Vector plot of velocity
      plot contour pressure    # Contour plot of pressure
      plot coloured contour pressure
      plot contour stream_function              # Contour plot of stream function
      plot coloured contour stream_function

end
```

7.1.18.2 2d Backward facing step, solved by Navier-Stokes, iterative solver

In order to get this example into your local directory use the command

```
sepgetex backwrd2_thpc
```

To run this example use:

```
sepmesh backwrd2_thpc.msh
view mesh
sepcomp backwrd2_thpc.prb
seppost backwrd2_thpc.pst
view results
```

The example is the standard backward facing step as described in Section (7.1.1). The only difference with the examples in (7.1.1) is that the system of equations is solved in a time-dependent way by pressure correction. Quadratic Taylor-Hood elements with linear pressure are used (type number 905 for the momentum equations and 906 for the pressure equation).
Compared to Section (7.1.1) only the problem file is essentially different and will be given here. All other files can be found in the sourceexam directory.

```
# backwrd2_thpc.prb
#
#  problem file for backward facing step
#  pressure-correction approach using Taylor-Hood elements
#  problem is stationary and non-linear, but is solved instationary
#
#  An iterative linear solver is applied
#  See Manual Examples Section 7.1.18
#
#  To run this file use:
#     sepcomp backwrd2_thpc.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
set warn off   ! suppress warnings
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
      t0        = 0                 # initial time
      dt        = 0.1               # time step
      tend      = 5                 # end time
      tout0     =  t0               # First time that a result is written
      toutend   =  tend             # End time for writing
      toutstep  = 5*dt            # In each 5^th time step the result is written
   integers
      outlet     = 21        # curve number for outlet boundary
      wall       = 25        # curve number for walls
      inflow     = 23        # curve number for inflow boundary
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=905                # Type number for Navier-Stokes, without swirl
                                # pressure correction method
                                # Taylor-Hood elements
                                # See Standard problems Section 7.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
                                # Only velocities are prescribed, not the
                                # pressures
      degfd1,degfd2=curves(c wall) # Fixed wall
```

```
      degfd1,degfd2=curves(c inflow) # inflow
      degfd2       =curves(c outlet) # Outstream boundary (v-component given)
                                # All not prescribed boundary conditions
                                # satisfy corresponding stress is zero

problem 2                 # See Users Manual Section 3.2.2
                                # solves the pressure equation

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=906                # Type number for pressure equation used in
                                # case of Navier-Stokes, pressure correction
                                # Taylor Hood approach
                                # See Standard problems Section 7.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
                                # The pressure is prescribed at the outlet
                                # In other boundaries we have dp/dn = 0
       curves(c outlet)         # Outlet boundary (p=0)

end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   storage_scheme = compact, problem = 1 # Non-symmetrical compact matrix for velocity
   storage_scheme = compact, symmetric, problem = 2  # symmetrical compact matrix for pressure
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c inflow), degfd1, quadratic   # The u-component of the velocity at
                                    # instream is quadratic
                                    # The rest of the vector is 0

end

#  Create pressure vector and set equal to 0

create vector, problem = 2
end

#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
```

```
      icoef5 = 1           # 5:  0: Stokes,  1: Picard
      icoef6 =  pressure   # 6:  iseqpress
      coef7  =  rho        # 7:  Density
      coef12 =  eta        #12:  Value of eta (viscosity)
end

#  Input for definition of pressure correction
#  In this case only defaults are used
#  See Users Manual Section 3.2.15

pressure_correction
   seq_vel_solver = 1
   seq_press_solver = 2
end


#  Definition of time integration
#  See Users Manual Section 3.2.15

time_integration
   method = euler_implicit         # Integration by the Euler implicit method
   tinit =  t0                     # Initial time
   tend =  tend                    # End time
   tstep =  dt                     # Time step
   toutinit =  tout0               # First time that a result is written
   toutend =  toutend              # End time for writing
   toutstep =  toutstep            # time steps for writing
   boundary_conditions = constant  # The boundary conditions do not depend on
                                   # time
   seq_boundary_conditions = 1     # Sequence number for the input of the
                                   # essential boundary conditions
   mass_matrix = constant          # Time-independent mass matrix

end

# Description of how the Navier-Stokes equations are solved
# See Users Manual, Section 3.2.22

navier_stokes
   method = pressure_correction       # solve by pressure correction
      seq_pressure_correction = 1     # sequence number of pressure correction
                                      # input
end

#  input for the linear solver (for both problems)
#  See Users Manual Section 3.2.8

solve, sequence_number = 1
   iteration_method = cg, preconditioner = ilu, print_level = 1
end
solve, sequence_number = 2
   iteration_method = cg, preconditioner = ilu, accuracy = 1d-5, print_level = 1
end

#
```

```
# Define the structure of the problem
# In this part it is described how the problem must be solved
#

structure                  # See Users Manual Section 3.2.3

  # Compute start vector for the flow by filling boundary conditions
   prescribe_boundary_conditions, velocity
   create_vector, pressure

  # Time loop
   start_time_loop

   # One time step to compute the velocity and the pressure,
   # using pressure correction

     navier_stokes

     output

   end_time_loop

end

end_of_sepran_input
```

7.1.19 Some examples of time dependent channel flow

In this section we show how the stationary channel flow can be solved as the limit of a time-dependent problem.
This Section show how the time-dependent Navier-Stokes equations can be solved in various ways. The problem itself is the simple channel flow described in Section (7.1.8). The only difference is that we add a time-derivative and start with a zero velocity, zero pressure at $t = 0$.
In the limit, for t large enough, the solution converges to steady state. Since the initial conditions do not match the boundary conditions we have an example of a transient.
The following examples are available

channelinstcrxx (7.1.19.1) Solves the time-dependent channel flow by the standard time integration and Crouzeix-Raviart elements (discontinuous pressure).

channelinstthxx (7.1.19.2) Solves the time-dependent channel flow by the standard time integration and Taylor-Hood elements (continuous pressure).

channelinstcsxx (7.1.19.3) Solves the time-dependent channel flow by the standard time integration and convection substepping.

channelinsthpc42 (7.1.18.1) Solves the channel flow using the Navier-Stokes equations and pressure-correction. A direct symmetric linear solver is used, and quadratic Taylor-Hood elements.

7.1.19.1 Time-dependent channel flow, Crouzeix-Raviart elements

In order to get this example into your local directory use the command

```
sepgetex channelinstcrxx
```

The following values for xx are available:

```
xx = 11   Quadratic triangles, with static condensation, type 900
xx = 12   Bi-linear quadrilaterals, type 900
xx = 13   Bi-quadratic quadrilaterals, with static condensation, type 900
xx = 21   Quadratic triangles, type 902
xx = 22   Bi-linear quadrilaterals, type 902
xx = 23   Bi-quadratic quadrilaterals, type 902
xx = 31   Quadratic triangles, with static condensation, type 901
```

The examples with type 900 use the penalty function formulation, the examples with type 901 and 902 the integrated approach.

To run this example use:

```
sepmesh channelinstcrxx.msh
view mesh
sepcomp channelinstcrxx.prb
seppost channelinstcrxx.pst
view results
```

The example is standard so no further explanation has to be given.
Only the problem file and a postprocessing file are given.

Note that the pressure computed in the three examples 11, 13 and 31 with static condensation is inaccurate. However, in the limit the pressure is correct.
The pressure in the other examples is also correct during the time-steps.
The examples with the integrated method require renumbering of the unknowns, preferably per level, in order to avoid singular matrices.

The input for program SEPCOMP is given in the following input file (channelinstcr11.prb):

```
#  channelinstcr11.prb
#
#
#  Instationary flow in channel (2D case)
#  Crouzeix-Raviart elements are used
#  Penalty function method
#
#  See Manual Standard Problems Section 7.1.10
#      Manual exams 7.1.19
#
#  Quadratic triangles with static condensation
#
#  To run this file use:
#     sepcomp channelinstcr11.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      inflow = 4                    # curve number of inflow boundary
      outlet = 2                    # curve number of outlet boundary
      wall   = 5                    # curve number of wall
   reals
      eps       = 1e-6              # penalty parameter for Navier-Stokes
      rho       = 1                 # density
      eta       = 0.01              # viscosity
      t0        = 0                 # initial time
      dt        = 0.1               # time step
      t1        = 5                 # end time
      tout0     = t0                # First time that a result is written
      toutend   = t1                # End time for writing
      toutstep  =  5*dt             # In each 5^th time step the result is written
      umax      = 1                 # maximum velocity at inflow
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=900                # Type number for Navier-Stokes, without swirl
                                # Penalty function approach
                                # Crouzeix-Raviart elements
                                # See Standard problems Section 7.1
```

```
   essbouncond                # Define where essential boundary conditions are
                              # given (not the value)
                              # See Users Manual Section 3.2.2
                              # Only velocities are prescribed, not the
                              # pressures
      degfd1,degfd2=curves(c wall) # Fixed wall
      degfd1,degfd2=curves(c inflow) # inflow
      degfd2       =curves(c outlet) # Outstream boundary (v-component given)
                              # All not prescribed boundary conditions
                              # satisfy corresponding stress is zero
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c inflow), degfd1, quadratic, max = umax   # The u-component of the velocity at
                                  # instream is quadratic
                                  # The rest of the vector is 0

end

#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 1           # 5:  1: Picard
      coef6  = eps         # 6:  Penalty function parameter eps
      coef7  = rho         # 7:  Density
      coef12 = eta         #12:  Value of eta (viscosity)
end

#  Definition of time integration
#  See Users Manual Section 3.2.15

time_integration
   method = euler_implicit         # Integration by the Euler implicit method
   tinit = t0                      # Initial time
   tend = t1                       # End time
   tstep = dt                      # Time step
   toutinit = tout0                # First time that a result is written
   toutend = toutend               # End time for writing
   toutstep = toutstep             # time steps for writing
   boundary_conditions = constant  # The boundary conditions do not depend on
                                   # time
   seq_boundary_conditions = 1     # Sequence number for the input of the
                                   # essential boundary conditions
   seq_coefficients = 1            # Sequence number for the coefficients
   mass_matrix = constant          # Time-independent mass matrix
   number_of_coupled_equations = 1 # There is only one equation
```

```
end

# Description of how the Navier-Stokes equations are solved
# See Users Manual, Section 3.2.22

navier_stokes
   method = standard                  # standard method
      seq_velocity = velocity        # sequence number of velocity vector
end

# compute pressure
# See Users Manual, Section 3.2.11

derivatives
   icheld=7          # icheld=7, pressure in nodes
                     # See Standard problems Section 7.1
end
#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#

structure                  # See Users Manual Section 3.2.3

  # Compute start vector for the flow by filling boundary conditions
   prescribe_boundary_conditions, sequence_number=1, vector=velocity

  # Time loop
   start_time_loop

   # One time step to compute the velocity
     time_integration, sequence_number = 1, velocity

   # Compute the pressure from the velocity
     derivatives, pressure

     output

   end_time_loop

end

end_of_sepran_input
```

The input for program SEPPOST is given in the following input file (channelinstcr11.pst):

```
#  channelinstcr11.pst
#
#  Instationary flow in channel (2D case)
#  Crouzeix-Raviart elements are used
#  Penalty function method
#
#  See Manual Standard Problems Section 7.1.10
#      Manual Examples Section 7.1.19
#
#  Quadratic triangles with static condensation
#
#
#  To run this file use:
#     seppost channelinstcr11.pst > channelinstcr11.out
#
#  Reads the files meshoutput and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2
#
#
#  compute the stream function
#  See Users Manual Section 5.2
#  store in stream_function

   compute stream_function = stream function velocity

#  Plot the results
#  See Users Manual Section 5.4

   time = (0, 10)
      plot vector velocity              # Vector plot of velocity
      plot contour pressure    # Contour plot of pressure
      plot coloured contour pressure
      plot contour stream_function              # Contour plot of stream function
      plot coloured contour stream_function

end
```

7.1.19.2 Time-dependent channel flow, Taylor-Hood elements

In order to get this example into your local directory use the command

```
sepgetex channelinstthxx
```

The following values for xx are available:

```
xx = 41   Linear elements, with static condensation, (mini-element)
xx = 42   Quadratic triangles
xx = 43   Bi-quadratic quadrilaterals
xx = 44   Linear elemens with extra point, no static condensation
```

All examples use type 903 and the integrated approach. Except for the mini element they all require renumbering of the unknowns, preferably per level, in order to avoid singular matrices.
In case of the mini element usually the convergence of the linear solver is better if no renumbering is applied.

To run this example use:

```
sepmesh channelinstthxx.msh
view mesh
sepcomp channelinstthxx.prb
seppost channelinstthxx.pst
view results
```

The example is standard so no further explanation has to be given.
Only the problem file and a postprocessing file are given.

Note that the pressure computed in examples 11 with static condensation is inaccurate. However, in the limit the pressure is correct.

The input for program SEPCOMP is given in the following input file (channelinstth41.prb):

```
#  channelinsthh41.prb
#
#
#  Instationary flow in channel (2D case)
#  Taylor-Hood elements are used
#  integrated method
#
#  See Manual Standard Problems Section 7.1.10
#      Manual examples Section 7.1.19
#
#  Linear triangles
#
#  To run this file use:
#     sepcomp channelinsthh41.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      inflow = 4                    # curve number of inflow boundary
      outlet = 2                    # curve number of outlet boundary
      wall   = 5                    # curve number of wall
   reals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
      t0        = 0                 # initial time
      dt        = 0.1               # time step
      t1        = 5                 # end time
      tout0     = t0                # First time that a result is written
      toutend   = t1                # End time for writing
      toutstep  =  5*dt             # In each 5^th time step the result is written
      umax      = 1                 # maximum velocity at inflow
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=903               # Type number for Navier-Stokes, without swirl
                               # Integrated approach
                               # Taylor-Hood elements
                               # See Standard problems Section 7.1
   essbouncond                 # Define where essential boundary conditions are
```

```
                              # given (not the value)
                              # See Users Manual Section 3.2.2
                              # Only velocities are prescribed, not the
                              # pressures
      degfd1,degfd2=curves(c wall) # Fixed wall
      degfd1,degfd2=curves(c inflow) # inflow
      degfd2       =curves(c outlet) # Outstream boundary (v-component given)
                              # All not prescribed boundary conditions
                              # satisfy corresponding stress is zero
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c inflow), degfd1, quadratic, max = umax   # The u-component of the
                                  # velocity at instream is quadratic
                                  # The rest of the vector is 0

end

#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 1           # 5:  1: Picard
      coef7  = rho         # 7:  Density
      coef12 = eta         #12:  Value of eta (viscosity)
end

#  Definition of time integration
#  See Users Manual Section 3.2.15

time_integration
   method = euler_implicit         # Integration by the Euler implicit method
   tinit = t0                      # Initial time
   tend = t1                       # End time
   tstep = dt                      # Time step
   toutinit = tout0                # First time that a result is written
   toutend = toutend               # End time for writing
   toutstep = toutstep             # time steps for writing
   boundary_conditions = constant  # The boundary conditions do not depend on
                                   # time
   seq_boundary_conditions = 1     # Sequence number for the input of the
                                   # essential boundary conditions
   seq_coefficients = 1            # Sequence number for the coefficients
   mass_matrix = constant          # Time-independent mass matrix
   number_of_coupled_equations = 1 # There is only one equation

end
```

```
# Description of how the Navier-Stokes equations are solved
# See Users Manual, Section 3.2.22

navier_stokes
   method = standard                  # standard method
      seq_velocity = velocity        # sequence number of velocity vector
end

# compute pressure
# See Users Manual, Section 3.2.11

derivatives
   icheld=7          # icheld=7, pressure in nodes
                     # See Standard problems Section 7.1
end
#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#

structure                   # See Users Manual Section 3.2.3

  # Compute start vector for the flow by filling boundary conditions
   prescribe_boundary_conditions, sequence_number=1, vector=velocity

  # Time loop
   start_time_loop

   # One time step to compute the velocity
     navier_stokes

   # Compute the pressure from the velocity
     derivatives, pressure

     output

   end_time_loop

end

end_of_sepran_input
```

7.1.19.3 Time-dependent channel flow, convection substepping

In order to get this example into your local directory use the command

```
sepgetex channelinstcs
```

The following values for xx are available:

```
xx = 42   Quadratic triangles, Taylor Hood
```

The examples are identical to the ones in Section (7.1.19.1) and (7.1.19.2). However, now convection substepping is applied.
At this moment the results are less accurate than the standard approach.

To run this example use:

```
sepmesh channelinstxx.msh
view mesh
sepcomp channelinstxx.prb
seppost channelinstxx.pst
view results
```

The example is standard so no further explanation has to be given.
Only the problem file is given.

The input for program SEPCOMP is given in the following input file (channelinstcs42.prb):

```
#  channelinstcs42.prb
#
#
#  Instationary flow in channel (2D case)
#  Taylor-Hood elements are used
#  integrated method, convection substepping
#
#  See Manual Standard Problems Section 7.1.10
#      Manual Examples Section 7.1.19
#
#  Quadratic triangles
#
#  To run this file use:
#     sepcomp channelinstcs42.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      inflow = 4                    # curve number of inflow boundary
      outlet = 2                    # curve number of outlet boundary
      wall   = 5                    # curve number of wall
      msteps = 2                    # number of substeps for
                                    # convective substepping
   reals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
      t0        = 0                 # initial time
      dt        = 0.1               # time step
      t1        = 5                 # end time
      tout0     = t0                # First time that a result is written
      toutend   = t1                # End time for writing
      toutstep  =  5*dt             # In each 5^th time step the result is written
      umax      = 1                 # maximum velocity at inflow
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=903               # Type number for Navier-Stokes, without swirl
                               # Integrated approach
                               # Taylor-Hood elements
```

```
                              # See Standard problems Section 7.1
   essbouncond                # Define where essential boundary conditions are
                              # given (not the value)
                              # See Users Manual Section 3.2.2
                              # Only velocities are prescribed, not the
                              # pressures
      degfd1,degfd2=curves(c wall) # Fixed wall
      degfd1,degfd2=curves(c inflow) # inflow
      degfd2       =curves(c outlet) # Outstream boundary (v-component given)
                              # All not prescribed boundary conditions
                              # satisfy corresponding stress is zero
   renumber levels (1,2), (3)
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c inflow), degfd1, quadratic, max = umax   # The u-component of the velocity at
                                  # instream is quadratic
                                  # The rest of the vector is 0

end

#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1
#  The first problem is the Stokes Problem, the second one
#  convection only

coefficients, sequence_number = 1     #  Stokes
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  0: Stokes
      coef7  = rho         # 7:  Density
      coef12 = eta         #12:  Value of eta (viscosity)
end

coefficients, sequence_number = 2     #  Convection step
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 2           # 5:  2: Newton, necessary to define u^n grad u^n
      coef7  = rho         # 7:  Density
      coef12 = eta         #12:  Value of eta (viscosity)
end

#  Definition of time integration
#  See Users Manual Section 3.2.15

time_integration
   method = euler_implicit         # Integration by the Euler implicit method
   tinit = t0                      # Initial time
   tend = t1                       # End time
```

```
   tstep = dt                       # Time step
   toutinit = tout0                 # First time that a result is written
   toutend = toutend                # End time for writing
   toutstep = toutstep              # time steps for writing
   boundary_conditions = constant   # The boundary conditions do not depend on
                                    # time
   seq_boundary_conditions = 1      # Sequence number for the input of the
                                    # essential boundary conditions
   seq_coefficients = 1             # Sequence number for the coefficients
                                    #   (Stokes part)
                                    # Convection part one higher
   mass_matrix = constant           # Time-independent mass matrix
   number_of_coupled_equations = 1 # There is only one equation

end

# Description of how the Navier-Stokes equations are solved
# See Users Manual, Section 3.2.22

navier_stokes
   method = convection_substepping    # convective equations explicit
      number_of_substeps = msteps     # number of substeps
      seq_time_integration = 1        # sequence number time integration input
      seq_velocity = velocity         # sequence number of velocity vector
end

# compute pressure
# See Users Manual, Section 3.2.11

derivatives
   icheld=7          # icheld=7, pressure in nodes
                     # See Standard problems Section 7.1
end
#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#

structure                  # See Users Manual Section 3.2.3

  # Compute start vector for the flow by filling boundary conditions
   prescribe_boundary_conditions, sequence_number=1, vector=velocity

  # Time loop
   start_time_loop

   # One time step to compute the velocity and pressure
     navier_stokes

   # Compute the pressure from the velocity
     derivatives, pressure

     output

   end_time_loop
```

```
end

end_of_sepran_input
```

7.1.20 Some examples of the use of the simple method

In this section we show the use of the simple method to solve the stationary Navier-Stokes equations. The following examples are available

(7.1.20.1) Solves the channel flow using the Stokes equations. Quadratic Taylor-Hood elements and the linear system is solved by standard SIMPLE (simple-gcr).

(7.1.20.2) Solves the 2d backward facing step using the Navier-Stokes equations. The non-linear iteration is done by Newton and the linear systems are solved by SIMPLE (simple-gcr). Quadratic Crouzeix-Raviart elements are applied.

7.1.20.1 Channel flow, solved by Stokes, SIMPLE solver

In order to get this example into your local directory use the command

```
sepgetex channelsimth41
```

To run this example use:

```
sepmesh channelsimth41.msh
view mesh
sepcomp channelsimth41.prb
```

The example is identical to the one in Section (7.1.8). Since the convective terms have no influence in this case only the linear Stokes equations are used. Taylor-Hood elements are used, with quadratic velocity and linear pressure.
The problem is standard. The only difference with the examples in Section (7.1.8) is that the linear problem is solved with the SIMPLE-GCR method. The sub-equations for velocity and pressure are solved by Conjugate gradients and an ILU preconditioner. The input for the mesh generator is not shown in this section.
No input file for the postprocessor is available, since the solution is trivial.

The input file for SEPCOMP is given by the file channelsimth41.prb:

```
#  channelsimth41.prb
#
#  problem file for 2d channel problem solved by simple iteration
#  Taylor Hood quadratic elements (linear pressure)
#
#  See Manual Standard Elements Section 7.1.20
#
#  To run this file use:
#     sepcomp channelsimth41.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
   vector_names
      velocity
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=903                # Type number for Navier-Stokes, without swirl
                                # integrated approach
                                # See Standard problems Section 7.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
     degfd1, degfd2,curves(c1) # Fixed under wall
     degfd1, degfd2,curves(c3) # Fixed side walls and instream boundary
     degfd1, degfd2,curves(c4) # inflow
     degfd2=curves(c2)          # Outstream boundary (v-component given)
                                # All not prescribed boundary conditions
                                # satisfy corresponding stress is zero
   reorder plast                # renumber the unknowns such that first we
                                # have all velocities and then all pressures
                                # Necessary for simple
end
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because the integral of the pressure over the boundary
#  is required
#
structure                  # See Users Manual Section 3.2.3
```

```
 # Compute the velocity
   prescribe_boundary_conditions, velocity
   solve_linear_system, velocity
 # Write the results to a file
   output
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   storage_scheme = simple, symmetric, incompressibility_sym
                              # The simple method is applied (iterative method)
                              # Momentum matrix is symmetrical
                              # G = D^T, i.e. gradient matrix is transpose of
                              # divergence matrix
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c4), degfd1, quadratic   # The u-component of the velocity at
                                   # instream is quadratic
                                   # The rest of the vector is 0

end

#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef7  = rho        # 7:  Density
      coef12 = eta        #12:  Value of eta (viscosity)
end

#  Input for the linear solver
#  Simple is applied, i.e. the overall method is GCR simple
#  Each of the sub-equations is solved by Conjugate Gradients,
#  with an ILU pre-conditioner

solve
   iteration_method = simple_gcr, preconditioning = ilu, print_level = 2//
      start = old_solution, accuracy = 1d-3
   sub_equation 1
      iteration_method = cg, preconditioning = ilu, print_level = 0, eps = 0.1
   sub_equation 2
      iteration_method = cg, preconditioning = ilu, print_level = 0, eps = 0.1
end
```

```
end_of_sepran_input
```

7.1.20.2 2d Backward facing step, solved by Navier-Stokes, SIMPLE solver

In order to get this example into your local directory use the command

```
sepgetex backwrdsim
```

To run this example use:

```
sepmesh backwrdsim.msh
view mesh
sepcomp backwrdsim.prb
seppost backwrdsim.pst
view results
```

The example is the standard backward facing step as described in Section (7.1.1). The only difference with the examples in (7.1.1) is that the system of equations is solved by the SIMPLE-GCR method. Crouzeix-Raviart elements with extended quadratic velocity and discontinuous linear pressure are used (type number 902).
Compared to Section (7.1.1) only the problem file is essentially different and will be given here. All other files can be found in the sourceexam directory.

```
#  backwrdsim.prb
#
#  problem file for backward facing step solved by simple iteration
#  Crouzeix-Raviart quadratic elements (linear pressure)
#
#  See Manual Standard Elements Section 7.1.20
#
#  To run this file use:
#     sepcomp backwrdsim.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
set warn off   ! suppress warnings
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      rho       = 1                 # density
      eta       = 0.01              # viscosity
      eps       - 0                 # compressibility
   integers
      lower_wall = 20        # curve number for lower wall
      outflow    = 21        # curve number for outflow boundary
      upper_wall = 22        # curve number for upper wall
      inflow     = 23        # curve number for inflow boundary
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=902                # Type number for Navier-Stokes, without swirl
                                # See Standard problems Section 7.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      curves(c lower_wall)      # Fixed under wall (velocity given)
      curves(c upper_wall)      # Fixed upper wall (velocity given)
      degfd2,curves(c outflow) # Outflow boundary (v-component 0)
      curves(c inflow)          # Inflow boundary (velocity given)

   reorder plast                # renumber the unknowns such that first we
                                # have all velocities and then all pressures
                                # Necessary for simple
end

#  Define the structure of the large matrix
```

```
#  See Users Manual Section 3.2.4
matrix
   storage_scheme = simple     # The simple method is applied (iterative method)
                               # Momentum matrix is not symmetrical
                               # G = D^T, i.e. gradient matrix is transpose of
                               # divergence matrix
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

   curves(c inflow), degfd1, quadratic   # The u-component of the velocity at
                                  # instream is quadratic
                                  # The rest of the vector is 0

end

#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 0           # 5:  Type of linearization (0=Stokes flow)
      coef6  =  eps        # 6:  Penalty function parameter eps
      coef7  =  rho        # 7:  Density
      coef12 =  eta        #12:  Value of eta (viscosity)
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations
   global_options, maxiter=8, accuracy=1d-3,print_level=2, lin_solver=1 //
      at_error return
   equation 1
      fill_coefficients 1
      change_coefficients
```

```
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end


#  Input for the linear solver
#  Simple is applied, i.e. the overall method is GCR simple
#  Each of the sub-equations is solved by Conjugate Gradients,
#  with an ILU pre-conditioner

solve
   iteration_method = simple_gcr, preconditioning = ilu, print_level = 1//
      start = old_solution, accuracy = 1d-3
   sub_equation 1
      iteration_method = cg, preconditioning = ilu, print_level = 0, eps = 0.1
   sub_equation 2
      iteration_method = cg, preconditioning = ilu, print_level = 0, eps = 0.1
end

# Define the structure of the problem
# In this part it is described how the problem must be solved

structure                  # See Users Manual Section 3.2.3

  # Compute start vector for the flow by filling boundary conditions
   prescribe_boundary_conditions, velocity

  # Compute the velocity, i.e. solve non-linear problem
   solve_nonlinear_system, velocity

  # Compute the pressure
   derivatives, pressure

  # Write the results to a file
   output
end

#  The pressure is computed as a derived quantity of the Navier-Stokes
#  equation
#  See Users Manual Section 3.2.11 and Standard Problems Section 7.1

derivatives, sequence_number = 1
   icheld = 7        # means compute pressure
end

end_of_sepran_input
```

7.1.21 Computation of shear stress in flow with constriction

We consider the flow in a constriction as described in Effect of constriction height on flow separation in a two-dimensional channel, by G.C. Layek and C. Midya, Communications in non-linear Science and Numerical Simulation 12, 2007, pp. 745-759. It concerns a straight channel with a restriction of cosine form as shown in Figure 7.1.21.1. At inflow we have a quadratic inflow profile with maximum velocity *umax*, the horizontal walls are no-slip and the out flow is parallel to the walls. In fact the flow is symmetrical and one could restrict one selves to one half of the region.
The example itself is more or less standard. The different thing with other examples is that it shows how the shear stress along the walls can be computed.
In order to get these examples into your local directory use the command

```
sepgetex constriction
```

To run this example use:

```
seplink constrictionmesh
constrictionmesh < constriction.msh
view mesh
sepcomp constriction.prb
seppost constriction.pst
view results
```

Since the boundary of the mesh is defined by a function, we need to add a function subroutine `funccv` and therefore main program constrictionmesh is used. The contents of the the files constrictionmesh.f and constriction.msh are self explaining and will not be repeated here.

Also the first part of the problem file `constriction.prb` is standard. New in this example is the computation of the shear stress. This is done in the structure block, where we first compute the stress tensor t, i.e. without the contribution of the pressure. The stress tensor always consists of 6 components per point. To get the stress vector along the boundary, the stress tensor must be multiplied by the normal vector. This is done in the statement

```
normal_stress = stress_vector stress, curves = (c wall1, c wall2),
```

where wall1 and wall2 are the curve numbers of both the walls. In order to get the components perpendicular to the wall (normal stress σ^{nn}) and tangential to the wall (shear stress σ^{nt}) we have to multiply the stress vector by the normal. This is done by

```
normal_stress = transform_to_normaldir normal_stress//
    curves = (c wall1, c wall2).
```

The first component refers to the normal stress the second one to the shear stress. In order to subtract the pressure from the normal stress, we have to store the normal stress in a vector with one degree of freedom per point. The following statements take care of this and do the subtraction

```
sigma_nn = extract normal_stress, degfd1
sigma_nn = sigma_nn - pressure  ! subtract the pressure
```

The Navier-Stokes equations are solved by the penalty function method (type 900) with bi-quadratic quadrilaterals.
The input file for SEPCOMP is given by the file constriction.prb:

```
#  constriction.prb
#
#
```

```
#  Example file for problem described in the paper:
#
#  Effect of constriction height on flow separation in a two-dimensional
#  channel, by G.C. Layek and C. Midya, Communications in non-linear Science
#  and Numerical Simulation 12, 2007, pp. 745-759
#
#  Crouzeix-Raviart elements are used
#  Penalty function method
#
#  See Manual Standard Problems Section 7.1.10
#      Manual exams 7.1.21
#
#  Quadratic quadrilaterals with static condensation
#
#  To run this file use:
#     sepcomp constriction.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      inflow = 4                    # curve number of inflow boundary
      outlet = 2                    # curve number of outlet boundary
      wall1 = 1                     # curve number of lower wall
      wall2 = 3                     # curve number of upper wall
   reals
      re        = 600               # Reynolds number
      eps       = 1e-6              # penalty parameter for Navier-Stokes
      rho       = 1                 # density
      umax      = 0.25              # inflow velocity
      eta       = 1/re              # viscosity
   vector_names
      velocity                      # velocity vector
      pressure                      # pressure
      stress                        # stress tensor in whole domain
      normal_stress                 # stress vector along walls
      sigma_nn                      # normal component of stress vector
                                    # including contribution of pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=900               # Type number for Navier-Stokes, without swirl
                               # Penalty function approach
                               # Crouzeix-Raviart elements
                               # See Standard problems Section 7.1
```

```
   essbouncond                # Define where essential boundary conditions are
                              # given (not the value)
                              # See Users Manual Section 3.2.2
                              # Only velocities are prescribed, not the
                              # pressures
      degfd1,degfd2=curves(c wall1) # Fixed wall
      degfd1,degfd2=curves(c wall2) # Fixed wall
      degfd1,degfd2=curves(c inflow) # inflow
      degfd2       =curves(c outlet) # Outstream boundary (v-component given)
                              # All not prescribed boundary conditions
                              # satisfy corresponding stress is zero
end

# Information about the matrix storage
#  See Users Manual Section 3.2.4

matrix
   storage_scheme = profile
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions
   curves(c inflow), degfd1, quadratic, max = umax   # The u-component of the
                                    # velocity at instream is quadratic
                                    # The rest of the vector is 0
end
#  Define the coefficients for the problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1
coefficients
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      icoef2 = 1           # 2:  type of constitutive equation (1=Newton)
      icoef5 = 1           # 5:  1: Picard
      coef6  = eps         # 6:  Penalty function parameter eps
      coef7  = rho         # 7:  Density
      coef12 = eta         #12:  Value of eta (viscosity)
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

# input for non-linear solver
```

```
# See Users Manual Section 3.2.9

nonlinear_equations
   global_options, maxiter=10, accuracy=1d-4,print_level=2, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

# compute pressure
# See Users Manual, Section 3.2.11

derivatives, sequence_number = 1
   icheld=7          # icheld=7, pressure in nodes
                     # See Standard problems Section 7.1
end

# compute stress
# See Users Manual, Section 3.2.11

derivatives, sequence_number = 2
   icheld=6          # icheld=6, stress in nodes
                     # See Standard problems Section 7.1
end

#
# Define the structure of the problem
# In this part it is described how the problem must be solved
#
structure                  # See Users Manual Section 3.2.3

  # Compute start vector for the flow by filling boundary conditions

   prescribe_boundary_conditions, vector=velocity

  # Compute the velocity, i.e. solve non-linear problem
   solve_nonlinear_system, velocity

   # Compute the pressure from the velocity
     derivatives, pressure

   # Compute the stress tensor from the velocity
     derivatives, stress

   # Compute the stress vector along the walls from the stress
     normal_stress = stress_vector stress, curves = (c wall1, c wall2)

   # Transform the stress vector into normal and tangential components
   # i.e. normal stress and shear stress
     normal_stress = transform_to_normaldir normal_stress//
         curves = (c wall1, c wall2)

   # Put the normal component into sigma_nn
```

```
     sigma_nn = extract normal_stress, degfd1
     sigma_nn = sigma_nn - pressure  ! subtract the pressure
     output

end
end_of_sepran_input
```

The other files can be found by sepgetex.

Figure 7.1.21.2 shows the shear stress along the lower wall.

7.2 The temperature dependent laminar flow of incompressible liquids (Boussinesq approximation)

7.2.1 Laminar Newtonian free convection flow by the penalty function method (coupled approach)

In this example we consider a free convection problem described by a Newtonian viscosity model. To get this example in your local directory use the command:

```
sepgetex bousscop
```

To run the example use the commands:

```
sepmesh bousscop.msh
view the plots
sepcompexe bousscop.prb
seppost bousscop.pst
view the plots
```

Consider a square container with different temperatures at left and right walls. The upper and lower walls are supposed to be isolated. Due to the temperature difference and the acceleration due to gravity, a circulating flow arises. The velocity at the boundaries is equal to zero (fixed walls). Figure 7.2.1.1 shows the region of definition as well as the curves and points defining the geometry.

This problem is a well known bench mark problem for free convection flows, see for example de Vahl Davis (1982).

The following boundary conditions are imposed:

All walls (C4 to C7): no-slip conditions ($\mathbf{v} = \mathbf{0}$)
Lower (C4) and upper (C6) wall: isolated ($\frac{\partial T}{\partial n}$)
Left wall (C7): $T = 1$
Right wall (C5): $T = 0$

The parameters used in this problem are: $\eta = 1$
$\rho = 1$
$\varepsilon = 10^{-8}$
$\omega = 0$
$c_p = 1$ $T_0 = T_1 = 0$
$\mathbf{f} = 0, f_T = 0$
Rayleigh number (Ra) $= 10^3$
Prandtl number (Pr) $= 0.71$
hence: $\beta = \frac{Ra}{9.81Pr}$, $\kappa = \frac{1}{Pr}$

The (coarse) mesh is generated by program SEPMESH using the following input file:

```
#  bousscop.msh
#
#  mesh file for 2d Boussinesq problem
#  coupled approach
#  See Manual Standard Elements Section 7.2.1
#
#  To run this file use:
#     sepmesh bousscop.msh
#
#  Creates the file meshoutput
```

```
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      step1 = 0.2   # Defines boundary layer at left-hand side
      step2 = 0.8   # Defines boundary layer at right-hand side
      x0    = 0     # x-coordinate left-hand wall
      x1    = 1     # x-coordinate right-hand wall
      y0    = 0     # y-coordinate lower wall
      y1    = 1     # y-coordinate upper wall
   integers
      nelm_layer = 3   # Number of elements in boundary layer
      nelm_width = 5   # Number of elements in vertical direction
      nelm_centr = 3   # Number of elements in horizontal direction
                       # between boundary layers
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=( x0, y0)       # Left under point
      p2=( x1, y0)       # Right under point
      p3=( x1, y1)       # Right upper point
      p4=( x0, y1)       # Left upper point
      p5=( step1, y0)    # Lower point left boundary layer
      p6=( step2, y0)    # Lower point right boundary layer
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=line2(p1,p5,nelm= nelm_layer,ratio=3,factor=0.8) # curve in lower wall
                                                          # left boundary layer
      c2=line2(p5,p6,nelm= nelm_centr)                    # curve in lower wall
                                                          # central part
      c3=line2(p6,p2,nelm= nelm_layer,ratio=1,factor=0.8) # curve in lower wall
                                                          # right boundary layer
      c4=curves(c1,c2,c3)                                 # lower boundary
      c5=line2(p2,p3,nelm= nelm_width)                    # left-hand wall
      c6=translate c4(p4,-p3)                             # upper wall
      c7=translate c5(p1,p4)                              # right-hand wall
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
                    # Quadratic triangles are used
      s1=quadrilateral4(c4,c5,-c6,-c7)    # See Users Manual Section 2.4.3
   plot                          # make a plot of the mesh
                                 # See Users Manual Section 2.2
end
```

The iteration process is carried out by starting with the Stokes solution, followed by one Picard iteration and followed by Newton iterations. The corresponding input file for program sepcomp is:

```
# bousscop.prb
set warn off
#
#  problem file for 2d Boussinesq problem
#  decoupled approach
#  problem is stationary and non-linear
#  The velocity and temperature are solved in a coupled way
#  See Manual Exams Section 7.2.1
#
#  To run this file use:
#     bousscop < bousscop.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps_penal =   1e-8       # penalty parameter
      rho       =   1          # density
      eta       =   1          # viscosity
      g         =   9.81       # acceleration of gravity
      Pr        =   0.71       # Prandtl number
      Ra        =   1e3        # Rayleigh number
      cp        =   1          # Heat capacity at constant temperature
      kappa     =  1/Pr        # Thermal conductivity kappa = 1/Pr
      beta      =  Ra/(g*Pr)   # volume expansion coefficient beta = Ra/(g Pr)

   integers
      veloc = 1      # sequence number velocity vector
      temp  = 2      # sequence number concentration vector
   vector_names
      velocity
      pressure
end
#
#  Define the type of problem to be solved
#
problem                     # See Users Manual Section 3.2.2
   types                    # Define types of elements,
                            # See Users Manual Section 3.2.2
      elgrp1 (type=420)     # Boussinesq by penalty function formulation
                            # See Manual Standard Elements Section 7.2
   essbouncond              # Define where essential boundary conditions are
                            # given (not the value)
                            # See Users Manual Section 3.2.2
      curves(c5,c7)         # Velocity and temperature prescribed
      degfd1,degfd2=curves(c4, c6)     # Velocity prescribed
end
```

```
#  Define the structure of the large matrix

matrix                       # See Users Manual Section 3.2.4
      storage_scheme = profile
      # The matrix is non-symmetrical and stored as profile matrix,
      # hence a direct solver is applied
end

#  Definition of the boundary conditions

essential boundary conditions       # See Users Manual Section 3.2.5
   curves (c7), degfd3=(value=1)    # All boundary conditions are zero, except
                                    # the temperature at wall c7
end

# input for non-linear solver

nonlinear_equations, sequence_number = 1  # See Users Manual Section 3.2.9
   global_options, maxiter=10, accuracy=1d-4,print_level=1, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

#  Define the coefficients for the problems
#  See Users Manual Section 3.2.6

coefficients, sequence_number=1  # First iteration
                                 # See Manual Standard problems Section 7.1
   elgrp1
      coef 1: value= eps_penal   # penalty function parameter
                                 # The pressure is of order 1000
      coef 2: value= rho         # density of fluid
      icoef3 = 0                 # type of linearization of convective terms
                                 # (0 = no convective terms, stokes flow)
                                 # 4: angular velocity of rotating system
      coef 5: value= beta        #  volume expansion coefficient
                                 # 6: reference temperature
      coef 7: value= cp          # 7: heat capacity at constant pressure
      coef 8: value= kappa       # 8: thermal conductivity (1/Prandtl)
                                 # 9: body force in x-direction
                                 #10: body force in y-direction
                                 #11: heat source per unit mass
      icoef12 = 1                # type of constitutive equation
                                 # (1=eta constant)
      coef 13: value= eta        # eta
                                 # 14: ct
                                 # 15: reference temperature T1
end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7
```

```
change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef3 = 1               # type of linearization of convective terms
                               # (1 = Picard)
end
change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef3 = 2               # type of linearization of convective terms
                               # (2 = newton)
end

#  The computed results are written
#  See Users Manual Section 3.2.13

output
   v1 = icheld=1   # pressure
end
end_of_sepran_input
```

Finally some post-processing actions are carried out by program SEPPOST using the following input file.

```
# bousscop.pst
#  Input file for postprocessing for Boussinesq example
#  Coupled approach
#  See Manual Standard Elements Section 7.2.1
#
#  To run this file use:
#     seppost bousscop.pst > bousscop.out
#
#  Reads the files meshoutput, sepcomp.inf and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2
#
#
#  print the vectors
#  See Users Manual Section 5.3
#
   print  velocity
   print  pressure
#
#  compute the stream function
#  See Users Manual Section 5.2
#
   compute stream function  velocity
#
#   plot the mesh
#  See Users Manual Section 5.4
#

   plot mesh

#
#   plot velocity vectors
```

```
#  See Users Manual Section 5.4
#
   plot vector  velocity, text = 'vector plot of velocity'
#
#  plot contour lines for the pressure, the stream function and
#  the temperature
#  See Users Manual Section 5.4
#
   plot contour  pressure, text = ' isobars'
   plot contour  stream_function
   plot contour  velocity, degfd=3, text = ' isotherms'

end
```

Figure 7.2.1.2 shows the mesh created by SEPMESH, Figure 7.2.1.3 the velocity vectors, Figure 7.2.1.4 the isobars, Figure 7.2.1.5 the stream lines and Figure 7.2.1.6 the isotherms.

7.2.2 Laminar Newtonian free convection flow by the penalty function method (decoupled approach)

This example is completely identical to the example treated in Section 7.2.1. However, instead of elements of type 420 containing both the velocity and temperature as unknowns, the temperature and velocity equations are solved separately.
To get this example in your local directory use the command:

```
sepgetex boussdec
```

To run the example use the commands:

```
sepmesh boussdec.msh
view the plots
seplink boussdec
boussdec < boussdec.prb
seppost boussdec.pst
view the plots
```

First a start vector is created with zero velocity and linear temperature ($T = 1 - x$) and then the velocity is solved using the standard Navier Stokes element of type 900 and the Boussinesq term $\rho \mathbf{g} \beta (T - T_0)$ with just generated temperature field as driving force. After that the temperature is solved by the standard convection diffusion equation with type number 900 and the just computed velocity in the convection terms. This process is repeated until convergence is achieved.
From the first step immediately Newton linearization is used.
Since the start vector depends on space it is necessary to use a function subroutine FUNC and hence the user must supply his own main program.
The following program boussdec.f might be used.

```
      program boussdec

!     --- Main program for decoupled Boussinesq equations
!         To link this program use:
!
!         seplink boussdec

      implicit none
      call sepcom(0)
      end

!     --- Function subroutine func is used to create an initial temperature

      function func ( ichois, x, y, z )
      implicit none
      integer ichois
      double precision func, x, y, z
      if ( ichois==1 ) then

!     --- ichois = 1, the temperature is set equal to 1-x

         func = 1 - x

      end if

      end
```

Because of the decoupled approach two problems have to be solved. Problem 1 corresponds to the momentum equations and problem 2 to the convection-diffusion equation for the temperature.
The solution consists of two vectors, the velocity (V1) and the temperature (V2).
The structure of the main program is organized by the input block STRUCTURE.
The following input file might be used to solve this problem:

```
# boussdec.prb
set warn off
#
#  problem file for 2d Boussinesq problem
#  decoupled approach
#  problem is stationary and non-linear
#  The velocity and temperature are solved in a decoupled way
#  See Manual Exams Section 7.2.2
#
#  To run this file use:
#     boussdec < boussdec.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#  General information:
#
#  Problem 1 refers to the velocity problem
#                   Navier-Stokes, laminar, Newtonian, isothermal
#            This is solved by quadratic elements
#  Problem 2 refers to the temperature problem
#                   Convection-diffusion
#            This is solved by quadratic elements
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps_penal =   1e-8       # penalty parameter
      rho       =   1          # density
      eta       =   1          # viscosity
      g         =   9.81       # acceleration of gravity
      Pr        =   0.71       # Prandtl number
      Ra        =   1e3        # Rayleigh number
      cp        =   1          # Heat capacity at constant temperature
      kappa     =  1/ Pr       # Thermal conductivity kappa = 1/Pr
      beta      =   Ra/( g* Pr)  # volume expansion coefficient beta = Ra/(g Pr)
      fy        =   g* beta    # body force in y-direction fy = rho g beta T

   integers
      veloc = 1      # sequence number velocity vector
      temp  = 2      # sequence number concentration vector
   vector_names
      velocity
      temperature
      pressure
end
#
#  Define the type of problem to be solved
```

```
problem  veloc              # See Users Manual Section 3.2.2
                            # Defines velocity problem
   types                    # Define types of elements,
                            # See Users Manual Section 3.2.2
      elgrp1 (type=900)     # Navier-Stokes by penalty function formulation
                            # See Manual Standard Elements Section 7.1
   essbouncond              # Define where essential boundary conditions are
                            # given (not the value)
                            # See Users Manual Section 3.2.2
      curves(c4 to c7)       # All velocities are prescribed
problem  temp               # See Users Manual Section 3.2.2
                            # Defines temperature problem
   types                    # Define types of elements,
                            # See Users Manual Section 3.2.2
      elgrp1 (type=800)     # Convection diffusion equation
                            # See Manual Standard Elements Section 3.1
   essbouncond              # Define where essential boundary conditions are
                            # given (not the value)
                            # See Users Manual Section 3.2.2
      curves (c5,c7)        # is prescribed
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#
structure                   # See Users Manual Section 3.2.3

 # create initial conditions, both for velocity and temperature
   create_vector velocity, problem = veloc, value = 0
   create_vector temperature, problem = temp, func = 1
 # Solve system of nonlinear equations
   solve_nonlinear_system, velocity
 # Compute the pressure as derived quantity
   pressure = derivatives ( velocity, seq_coef = veloc, icheld = 7 )
 # Write the results to a file
   output
end

#  Define the structure of the large matrix

matrix                      # See Users Manual Section 3.2.4
   storage_scheme=profile, problem= veloc
                                # The velocity matrix is non-symmetrical and
                                # stored as profile matrix,
                                # hence a direct solver is applied
   storage_scheme=profile, problem= temp
                                # The temperature matrix is non-symmetrical
                                # and stored as profile matrix,
                                # hence a direct solver is applied
end

# input for non-linear solver

nonlinear_equations     # See Users Manual Section 3.2.9
   number_of_coupled_equations = 2     # The velocity and temperature
```

```
                                    # equation are solved as a set of two
                                    # equations
   global_options print_level=2     # Print information about the convergence
   equation  veloc                  # Velocity equation
      fill_coefficients= veloc      # Information about the coefficients
   equation  temp                   # Temperature equation
      fill_coefficients= temp       # Information about the coefficients
end

#  Define the coefficients for the problems
#  See Users Manual Section 3.2.6

coefficients, sequence_number= veloc, problem =  veloc
                               # First problem (Navier-Stokes)
                               # See Manual Standard problems Section 7.1
   elgrp1
     icoef2 = 1                # type of constitutive equation (1=eta constant)
     icoef5 = 2                # type of linearization of convective terms
                               # (2 = newton linearization)
      coef 6: value= eps_penal # penalty function parameter
                               # The pressure is of order 1000
      coef 7: value= rho       # density of fluid
      coef10= old solution  temperature//
              coef= fy         # body force in y-direction (fy T)
     coef12: value= eta        # eta
end
coefficients, sequence_number= temp, problem =  temp
                               # Second problem (Convection-diffusion)
                               # See Manual Standard problems Section 3.1
   elgrp1
      coef6 =  kappa           # thermal conductivity
      coef9 = coef 6           # thermal conductivity
      coef12= old solution  velocity//
         degree of freedom 1   # u-velocity from Navier-Stokes
      coef13= old solution  velocity//
         degree of freedom 2   # v-velocity from Navier-Stokes
end
```

In this case the input file for program SEPPOST must also be adapted, however, the resulting pictures are exactly the same.

```
# boussdec.pst
#  Input file for postprocessing for Boussinesq example
#  Decoupled approach
#  See Manual Standard Elements Section 7.2.2
#
#  To run this file use:
#     seppost boussdec.pst > boussdec.out
#
#  Reads the files meshoutput, sepcomp.inf and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2
#
#
#  print the vectors
```

```
#  See Users Manual Section 5.3
#
   print  velocity
   print  temperature
   print  pressure
#
#   compute the stream function
#  See Users Manual Section 5.2
#
   compute stream function  velocity
#
#   plot velocity vectors
#  See Users Manual Section 5.4
#
   plot vector  velocity
#
#  plot contour lines for the pressure, the stream function and
#  the temperature
#  See Users Manual Section 5.4
#
   plot contour  pressure
   plot contour  stream_function
   plot contour  temperature

end
```

7.2.3 Time-dependent laminar Newtonian free convection flow by the penalty function method

This example is identical to the example treated in Section 7.2.2 extended with a time-derivative. To get this example in your local directory use the command:

```
sepgetex bousstim
```

To run the example use the commands:

```
sepmesh bousstim.msh
view the plots
seplink bousstim
bousstim < bousstim.prb
seppost bousstim.pst
view the plots
```

Since the problem is time-dependent we need an initial condition. At $t = 0$ we have the following initial conditions:

$$\mathbf{u} = \mathbf{0} \tag{7.2.3.1}$$

$$T = 1 - x \tag{7.2.3.2}$$

In each time step we first compute the velocity using the implicit Euler method and the Temperature at the previous time level. After that the Temperature is solved by implicit Euler using the velocity at the new time level. This approach is semi-implicit, since it is implicit per equation and uses the last computed values. In order to make it fully implicit it would be necessary to iterate per time-step. The present approach is known under the name Silechi.
Since the initial vector depends on space it is necessary to use a function subroutine FUNC and hence the user must supply his own main program.
The following program bousstim.f might be used.

```
      program bousstim

!     --- Main program for time-dependent Boussinesq equations
!         To link this program use:
!
!         seplink bousstim

      implicit none
      call sepcom(0)
      end

!     --- Function subroutine func is used to create an initial temperature

      function func ( ichois, x, y, z )
      implicit none
      integer ichois
      double precision func, x, y, z
      if ( ichois==1 ) then

!     --- ichois = 1, the temperature is set equal to 1-x

         func = 1 - x
```

```
      end if

      end
```

Because of the decoupled approach two problems have to be solved. Problem 1 corresponds to the momentum equations and problem 2 to the convection-diffusion equation for the temperature.
To show how derivative quantities can be solved in a time loop, also the pressure is computed in each time-step.
The solution consists of three vectors, the velocity (V1), the temperature (V2) and the pressure (V3).
The structure of the main program is organized by the input block STRUCTURE. However, in this particular example it is possible to skip this structure block, since it corresponds completely to the default block.
In this example all quantities are written at each time-level including $t = 0$. As a consequence the pressure must be initialized at $t = 0$.
The following input file might be used to solve this problem:

```
# bousstim.prb
set warn off
#
#  problem file for 2d time-dependent Boussinesq problem
#  problem is instationary and non-linear
#  The velocity and temperature in each time step are solved in a decoupled way
#  See Manual Exams Section 7.2.3
#
#  To run this file use:
#     bousstim < bousstim.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#  General information:
#
#  Problem 1 refers to the velocity problem
#                   Navier-Stokes, laminar, Newtonian, isothermal
#            This is solved by quadratic elements
#  Problem 2 refers to the temperature problem
#                   Convection-diffusion
#            This is solved by quadratic elements
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      eps_penal =   1e-8       # penalty parameter
      rho       =   1          # density
      eta       =   1          # viscosity
      g         =   9.81       # acceleration of gravity
      Pr        =   0.71       # Prandtl number
      Ra        =   1e3        # Rayleigh number
      cp        =   1          # Heat capacity at constant temperature
      kappa     =   1/Pr       # Thermal conductivity kappa = 1/Pr
      beta      =   Ra/(g*Pr)  # volume expansion coefficient beta = Ra/(g Pr)
      fy        =   g* beta    # body force in y-direction fy = rho g beta T
      rho_cp    =   rho*cp     # rho * cp
```

```
   integers
      veloc = 1      # sequence number velocity vector
      temp  = 2      # sequence number temperature vector
   vector_names
      velocity
      temperature
      pressure
end
#
#  Define the type of problem to be solved
problem  veloc              # See Users Manual Section 3.2.2
                            # Defines velocity problem
   types                    # Define types of elements,
                            # See Users Manual Section 3.2.2
      elgrp1 (type=900)     # Navier-Stokes by penalty function formulation
                            # See Manual Standard Elements Section 7.1
   essbouncond              # Define where essential boundary conditions are
                            # given (not the value)
                            # See Users Manual Section 3.2.2
      curves(c4 to c7)      # All velocities are prescribed
problem  temp               # See Users Manual Section 3.2.2
                            # Defines temperature problem
   types                    # Define types of elements,
                            # See Users Manual Section 3.2.2
      elgrp1 (type=800)     # Convection diffusion equation
                            # See Manual Standard Elements Section 3.1
   essbouncond              # Define where essential boundary conditions are
                            # given (not the value)
                            # See Users Manual Section 3.2.2
      curves (c5,c7)        # is prescribed
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#
structure                   # See Users Manual Section 3.2.3

 # create initial conditions, both for velocity and temperature
   create_vector velocity, problem veloc, value = 0
   create_vector temperature, problem, temp, func = 1
 # Solve time-dependent problem
   solve_time_dependent_problem velocity, sequence_number=1
 # Write the results to a file
   output
end

#  Define the structure of the large matrix

matrix                      # See Users Manual Section 3.2.4
   storage_scheme=profile, problem= veloc
                               # The velocity matrix is non-symmetrical and
                               # stored as profile matrix,
                               # hence a direct solver is applied
   storage_scheme=profile, problem= temp
```

```
                                # The temperature matrix is non-symmetrical
                                # and stored as profile matrix,
                                # hence a direct solver is applied
end

#  Define the coefficients for the problems
#  See Users Manual Section 3.2.6

coefficients, sequence_number= veloc, problem =  veloc
                               # First problem (Navier-Stokes)
                               # See Manual Standard problems Section 7.1
   elgrp1
     icoef2 = 1                # type of constitutive equation (1=eta constant)
     icoef5 = 2                # type of linearization of convective terms
                               # (2 = newton linearization)
      coef 6: value= eps_penal # penalty function parameter
                               # The pressure is of order 1000
      coef 7: value= rho       # density of fluid
      coef10= old solution  temperature//
              coef= fy         # body force in y-direction (fy T)
     coef12: value= eta        # eta
end
coefficients, sequence_number= temp, problem =  temp
                               # Second problem (Convection-diffusion)
                               # See Manual Standard problems Section 3.1
   elgrp1
      coef6 =  kappa           # thermal conductivity
      coef9 = coef 6           # thermal conductivity
      coef12= old solution  velocity//
         degree of freedom 1   # u-velocity from Navier-Stokes
      coef13= old solution  velocity//
         degree of freedom 2   # v-velocity from Navier-Stokes
      coef17: value= rho_cp    # rho cp
end
#
# Define the time integration
#  See Users Manual Section 3.2.15
#
time_integration, sequence_number = 1
   method = euler_implicit         # euler implicit time integration
   tinit = 0                       # t_0
   tend  = 0.5                     # end time
   tstep = 0.05                    # time step
   toutinit = 0                    # start writing at t=0
   toutend  = 0.5
   toutstep = 0.05
   boundary_conditions = initial_field
   seq_coefficients = 1, 2
   seq_output = 1
   mass_matrix = constant
   number_of_coupled_equations = 2
   equation 1
      derivatives, seq_deriv=1, problem= veloc, seq_coef= veloc//
                   vector= pressure
end
```

```
# The pressure is computed as a derived quantity of the Navier-Stokes
# equation
#  See Users Manual Section 3.2.11

derivatives, sequence_number = 1
   icheld = 7                  # Compute the pressure
                               # See Manual Standard problems Section 7.1
end
```

In this case the input file for program SEPPOST must also be adapted, however, the resulting pictures are exactly the same.

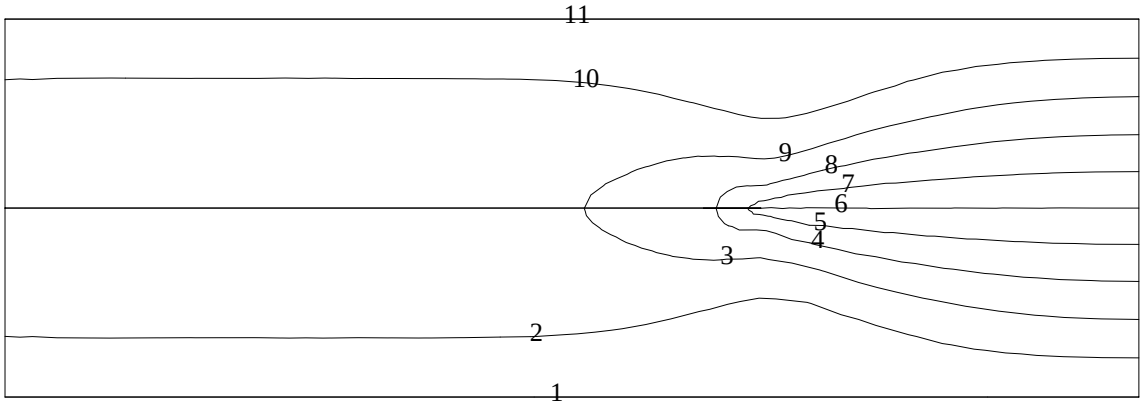
```
# bousstim.pst
#  Input file for postprocessing for time-dependent Boussinesq example
#  See Manual Standard Elements Section 7.1.5
#
#  To run this file use:
#     seppost bousstim.pst > bousstim.out
#
#  Reads the files meshoutput, sepcomp.inf and sepcomp.out
#
postprocessing                    # See Users Manual Section 5.2
#
#   compute the stream function
#  See Users Manual Section 5.2
#
   compute stream function  velocity
#
#  Define time loop for postprocessing
#  See Users Manual Section 5.5
#
   time = (0, 10)

#   print vectors
#   See Users Manual Section 5.3
#
    print  velocity
    print  pressure
    print  temperature
#   plot velocity vectors
#   See Users Manual Section 5.4
      plot vector  velocity, factor=.05
#
#  plot contour lines of the pressure
#  See Users Manual Section 5.4
#
      plot contour  temperature
      plot coloured contour  temperature, nlevel=21, mincolour=51
#
#  plot contour lines of the stream function
#  See Users Manual Section 5.4
#
      plot contour  stream_function
      plot coloured contour  stream_function, nlevel=21, mincolour=51
```

```
#
#  plot isotherms
#  See Users Manual Section 5.4
#
      plot contour  pressure
      plot coloured contour  pressure
#
#  end time loop
end
```

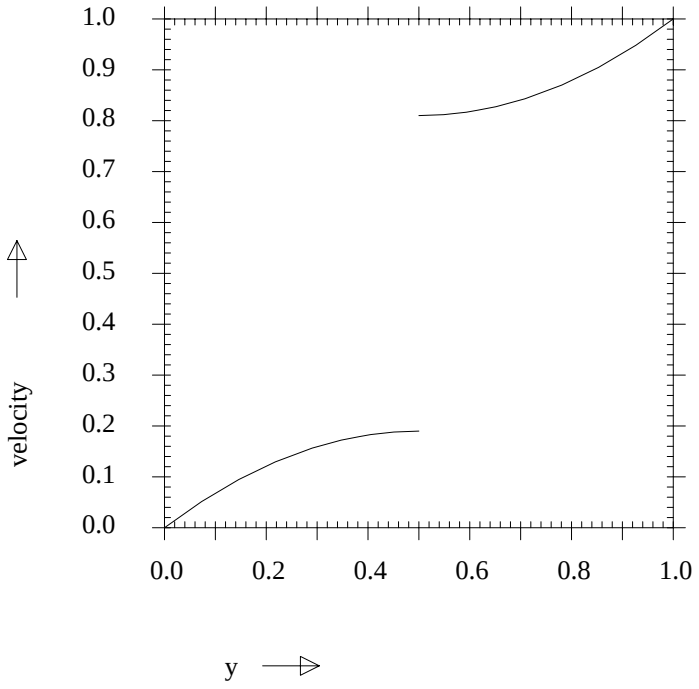
7.3 Turbulent flow

7.3.1 The isothermal turbulent flow of incompressible liquids according to Boussinesq's hypothesis

This section is under preparation

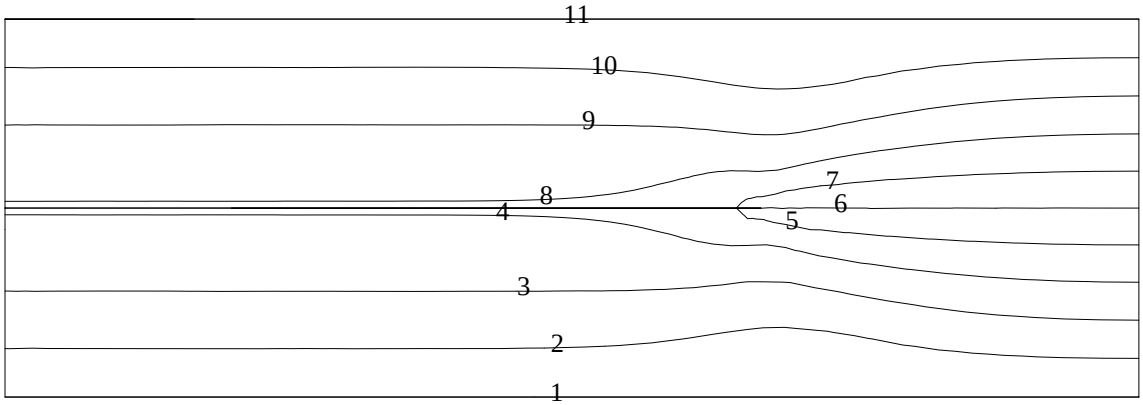


[height=6cm]

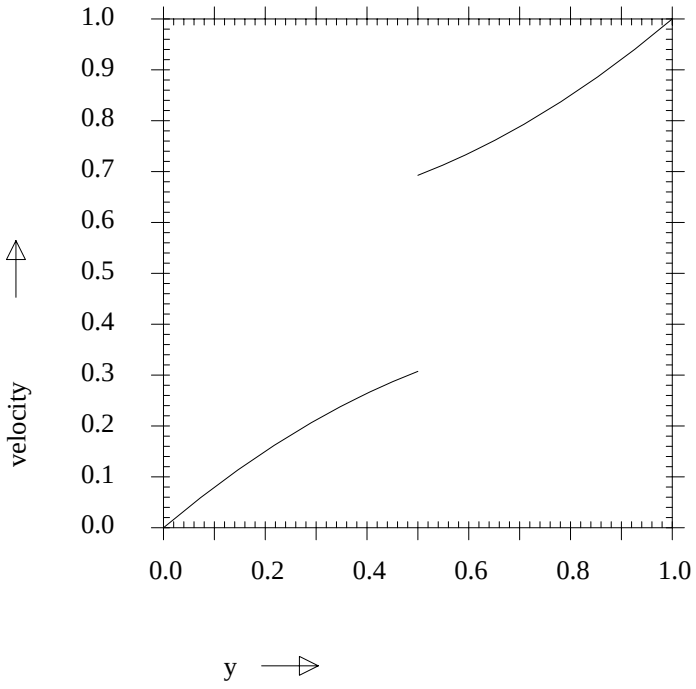


[height=6cm]

Figure 7.1.15.3: Contour plot and vertical cross section at $x = 0.5$ of the horizontal velocity field for $c_t = 0$. The fault friction is zero, which makes the fault free-slip. Beyond $x = 2$, the fault disappears and the flow develops into a Couette flow.

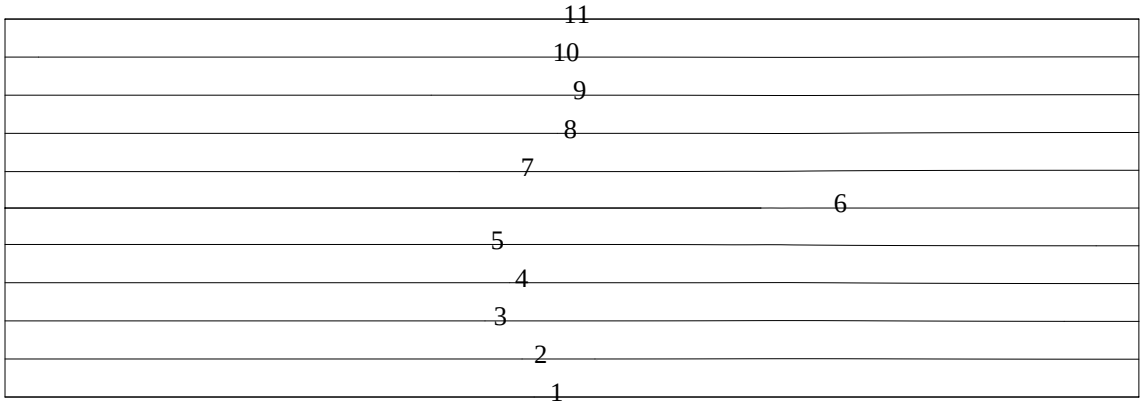


[height=6cm]

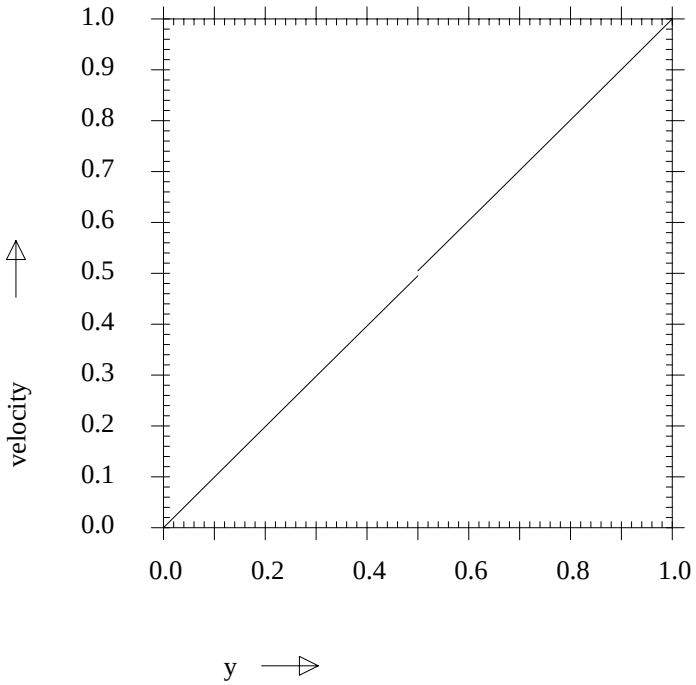


[height=6cm]

Figure 7.1.15.4: Contourplot and vertical cross section at $x = 0.5$ of the horizontal velocity field for $c_t = 1$. The fault friction is low. The relative displacement between top- and bottom boundary is divided between internal deformation and slip over the fault.



[height=6cm]



[height=6cm]

Figure 7.1.15.5: Contourplot and vertical cross section at $x = 0.5$ of the horizontal velocity field $c_t = 1$. The fault friction is high. The relative displacement over the fault is almost zero.

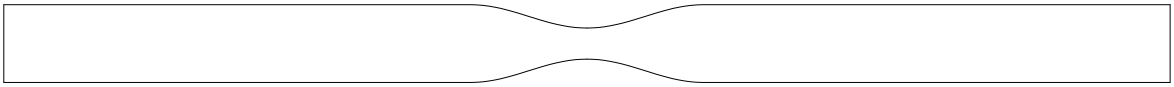


Figure 7.1.21.1: Channel with constriction

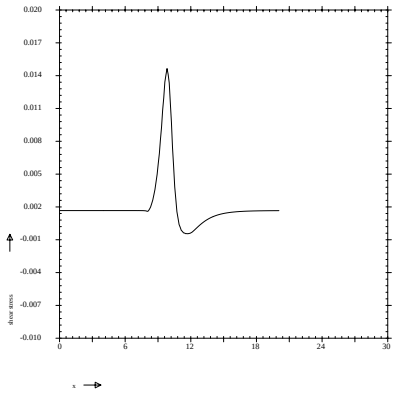


Figure 7.1.21.2: Shear stress along lower wall

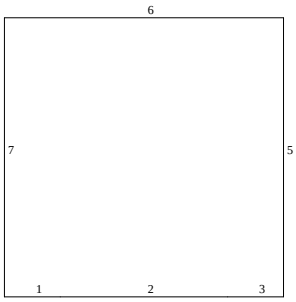


Figure 7.2.1.1: Definition of region for free convection flow

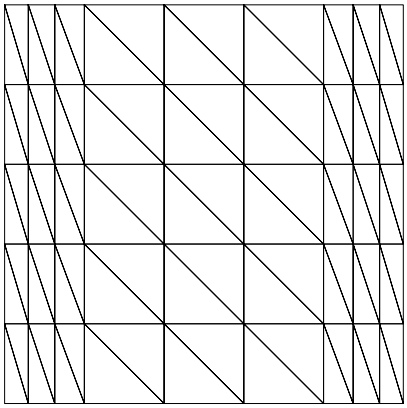
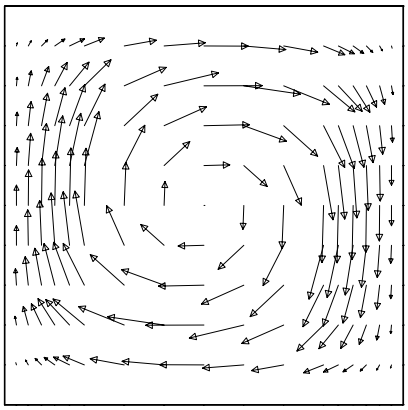
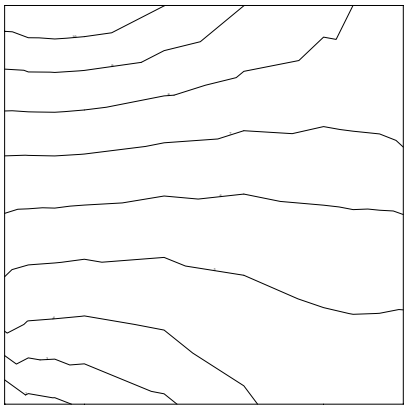


Figure 7.2.1.2: Mesh for free convection flow



vector plot of velocity

Figure 7.2.1.3: Velocities for free convection flow



isobars

Figure 7.2.1.4: Isobars for free convection flow

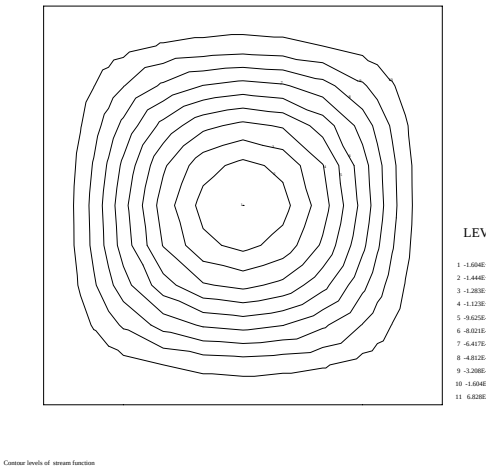


Figure 7.2.1.5: Stream lines for free convection flow

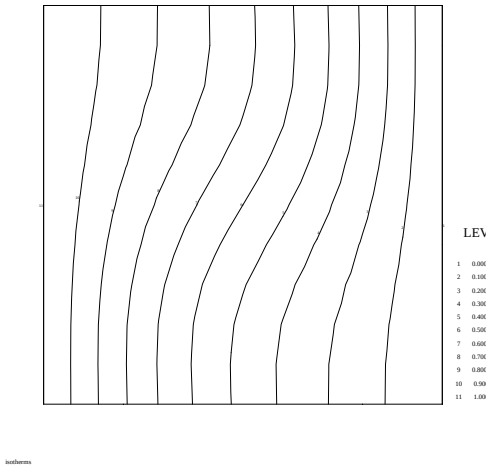


Figure 7.2.1.6: Isotherms for free convection flow

7.4 Methods to compute solid-fluid interaction

7.4.1 A very simple example of the fictitious domain method, a static solid in a fluid

To show how the fictitious domain method works, we start with a very simple 2d example of a non-moving solid in a fluid. Of course the method is meant for time-dependent problems, but this example shows the behavior of the method.
To get this example into your local directory use:

```
sepgetex fict_domain01
```

and to run it use:

```
sepmesh fict_domain01.msh
sepcomp fict_domain01.prb
seppost fict_domain01.pst
```

After the first and last step you may view the results using sepview.

Consider the following case of a small solid in a fluid as sketched in Figure 7.4.1.1

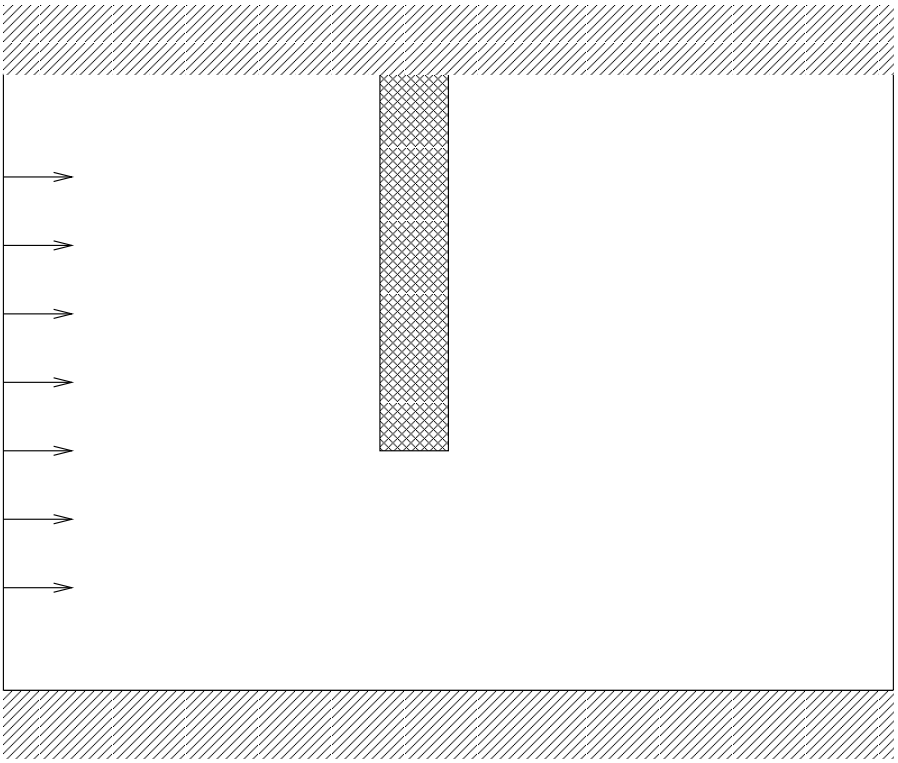


Figure 7.4.1.1: Sketch of region with obstacle

The solid is not moving and actually it may be considered as an obstacle, hence the velocity at the boundary of the solid is zero.
Although we must define structural elements on the solid, these elements are not essential for the computation. They just provide us a way to define the Lagrange multipliers.
On each boundary of the solid we put fictitious elements, except on the upper side, where we have a closed wall with a no-slip boundary condition. As a consequence it is not allowed to put a fictitious unknown on that boundary.

We consider a uniform flow at the left-hand side and we expect the fluid to flow around the obstacle. The input files for this problem are given by

```
# fict_domain01.msh
#
#  Mesh for 2d fictitious domain example
#  The problem considered here is that of a fixed small obstacle in the fluid
#
#  Mark that the mesh consists of two separate parts
#
#  See Manual Examples Section 7.4.1
#
#  To run this file use:
#     sepmesh fict_domain01.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      # Fluid mesh
      x_left = 0               # Left-hand side x-coordinate of fluid domain
      x_right = 3              # Right-hand side x-coordinate of fluid domain
      y_low = 0                # Lower y-coordinate of fluid domain
      y_top = 3                # Upper y-coordinate of fluid domain
      # Structure mesh
      x_left_obs = 1.4         # Left-hand side x-coordinate of obstacle
      x_right_obs = 1.45       # Right-hand side x-coordinate of obstacle
      y_low_obs = 1.5          # Lower y-coordinate of obstacle
   integers
      # Fluid mesh
      nelm_hor = 10            # number of elements in horizontal direction
      nelm_vert = 5            # number of elements in vertical direction
      # Structure mesh
      nelm_hor_obs = 2         # number of elements in horizontal direction
      nelm_vert_obs = 5        # number of elements in vertical direction
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2

      # Fluid mesh
         p1 = ( x_left,  y_low)       # Left under point
         p2 = ( x_right,  y_low)      # Right under point
         p3 = ( x_right,  y_top)      # Right upper point
         p4 = ( x_left,  y_top)       # Left upper point
      # Structure mesh
         p11 = ( x_left_obs,  y_low_obs)      # Left under point
         p12 = ( x_right_obs,  y_low_obs)     # Right under point
```

```
         p13 = ( x_right_obs,  y_top)         # Right upper point
         p14 = ( x_left_obs,  y_top)          # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # In the fluid domain quadratic elements are used
                    # In the solid domain linear elements are used
      # Fluid mesh
         c1 = line2 ( p1, p2, nelm =  nelm_hor )       # lower wall
         c2 = line2 ( p2, p3, nelm =  nelm_vert )      # outflow boundary
         c3 = line2 ( p3, p4, nelm =  nelm_hor )       # upper wall
         c4 = line2 ( p4, p1, nelm =  nelm_vert )      # inflow boundary
      # Structure mesh
         c11 = line ( p11, p12, nelm =  nelm_hor_obs )   # lower part
         c12 = line ( p12, p13, nelm =  nelm_vert_obs )  # right-hand side
         c14 = line ( p13, p14, nelm =  nelm_hor_obs )   # upper part
         c13 = line ( p14, p11, nelm =  nelm_vert_obs )  # left-hand side
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      # Fluid mesh
         s1 = rectangle6(c1,c2,c3,c4)     # Bi-quadratic quadrilaterals
      # Structure mesh
         s2 = rectangle5(c11,c12,c14,c13) # Bi-linear quadrilaterals
#
#  Connect surfaces with element groups
#
   meshsurf         # See Users Manual Section 2.2
      # Fluid mesh
        selm1=s1
      # Structure mesh
        selm2=s2
   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2
end

# fict_domain01.prb
#
#  Problem file for 2d fictitious domain example
#  The problem considered here is that of a fixed obstacle in the fluid
#
#  See Manual Examples Section 7.4.1
#
#  To run this file use:
#     sepcomp fict_domain01.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
```

```
constants           # See Users Manual Section 1.4
   reals
      rho       = 1000      # density of fluid
      eps_penal = 1e-6      # parameter eps for penalty function method
      eta       = 4e-3      # dynamic viscosity
      E         = 1e6       # Young's modulus
      nu        = 0.45      # Poisson's ratio
   vector_names
      velocity
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      # Fluid problem
      elgrp1=900                # Type number for Navier-Stokes, without swirl
      # Structure problem
      elgrp2 = 250              # Type number for elasticity problem

   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2

      # Structure problem
      surfaces(s2)                          # no movement at all

      # Fluid problem
      curves(c4)                            # inlet
      degfd2 = curves(c2)                   # outlet, only tangential velocity
      curves(c1)                            # no-slip bottom wall
      curves(c3)                            # no-slip top wall

   fictitious_unknowns          # Define type of elements to be used for
                                # ficititous domain method
      fictgrp 1 = type = 921    # See Users Manual Section 3.2.2 and
                                # Standard Problems Section 7.4

   fictitious_elements          # Define where the ficitious elements are
                                # positioned and the corresponding structure
                                # and fluid elements
      felm1 = curves = (c11 to c13), structure_group = 2, fluid_groups = 1//
         multiplier_shape=1

end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions

  degfd1 = curves(c4) value=1d-3   # The u-component of the velocity at
```

```
                                # inflow is constant
                                # The rest of the vector is 0

end

#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Sections 7.1, 5.1

coefficients, problem=1

   # Fluid problem
   elgrp1 ( nparm=20 )     # The coeffs are defined by 20 parameters
   icoef1 = 0              # Theta-method for time integration
   icoef2 = 1              # type of constitutive equation (1=Newton)
   icoef5 = 0              # Type of linearization (0=Stokes flow)
   coef6  =  eps_penal     # Penalty function parameter eps
   coef7  =  rho           # Density
   coef12 =  eta           #12:  Value of eta (dynamic viscosity)

   # Structure problem
   elgrp2 ( nparm=45 )     # The coeffs are defined by 45 parameters
   icoef2 = 0              # plane stress
   coef6  =  E             # Young's modulus (isotropic)
   coef7  =  nu            # Poisson's ratio (isotropic)


end
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This part is not necessary since what is used here is actually the default
#
structure                   # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_bounday_conditions, velocity
   solve_linear_system, velocity
 # Write the results to a file
   output
end

#  fict_domain01.pst
#  Input file for postprocessing for 2d fictitious domain example
#  The problem considered here is that of a fixed obstacle in the fluid
#
#  See Manual Examples Section 7.4.1
#
#
#  To run this file use:
#     seppost fict_domain01.pst > fict_domain01.out
#
#  Reads the files meshoutput and sepcomp.out
#
#
#  Define some general constants
#
```

```
postprocessing                    # See Users Manual Section 5.2

#  Plot the results
#  See Users Manual Section 5.4

   plot vector velocity              # Vector plot of velocity

end
```

Figure 7.4.1.2 shows the curve numbers used in this example and Figure 7.4.1.3 the corresponding mesh.

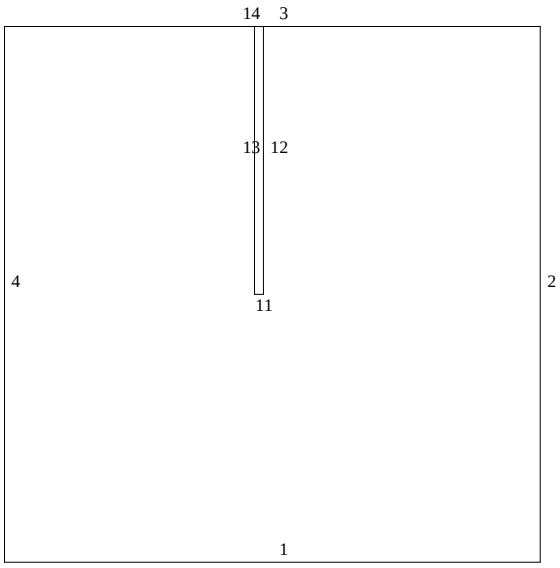


Figure 7.4.1.2: Curves for the solid in the fluid

Finally Figure 7.4.1.4 shows the computed velocity field.

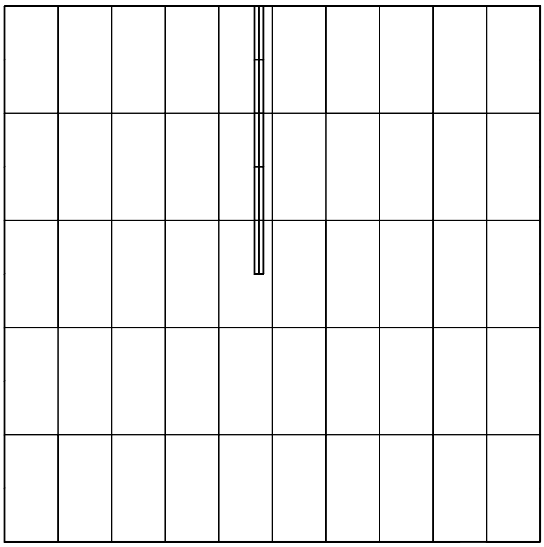


Figure 7.4.1.3: Mesh for the solid in the fluid

7.4.2 A simple Fluid domain deformation problem (weak coupling)

In this example we consider an elastic compressible solid, that deforms due to a time-dependent load on a part of the surface. The displacement acts as force (boundary condition) for a fluid flowing over the common interface. Due to the (large) displacement of the solid the region of the fluid is also changed.
In this example we show a weak coupling, which means that the fluid does not influence the solid, but the solid influences the fluid with boundary conditions and deformation of the domain.
To get this example into your local directory use:

```
sepgetex domain_def
```

and to run it use:

```
sepmesh domain_def.msh
seplink domain_def
domain_def < domain_def.prb
seppost domain_def.pst
```

After the first and last step you may view the results using sepview.

In this example, a fluid domain Ω^f and a connecting solid domain Ω^s with a common interface Γ_{fs} are considered (see Figure 7.4.2.1). In the fluid domain, a horizontal parabolic velocity is prescribed on the left boundary. The upper boundary is a wall and the left-hand boundary acts as outflow. The clamped part of the interface can be considered as a no-slip boundary for the fluid.
The solid is clamped on its boundaries apart from the common interface Γ_{fs}. On this interface, a time dependent force $f = f(x,t)$ is exerted on the solid in normal direction. This force is chosen as:

$$f(x,t) = 75 \cdot (\frac{1}{2} - \frac{1}{2}\cos(\frac{\pi}{3}x - \frac{2\pi}{3})) \cdot \sin(t) \cdot (\frac{1}{6}x - \frac{1}{3}) \qquad x \in \Gamma^{fs}. \tag{7.4.2.1}$$

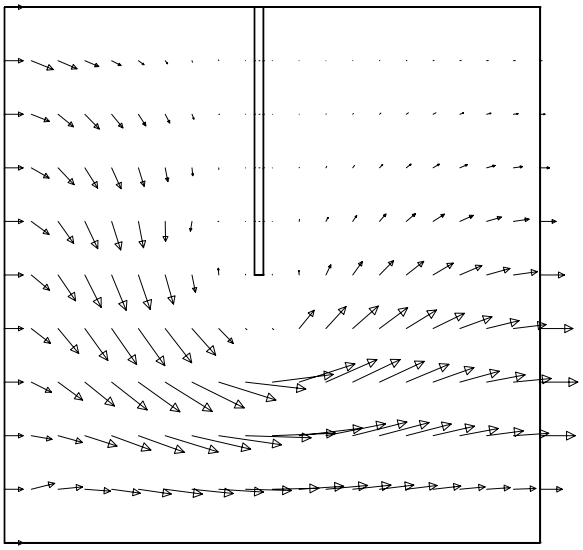


Figure 7.4.1.4: Velocity field for the solid in the fluid problem

Due to this time-dependent force, the solid will deform, and therefore the connected fluid domain will deform as well. In this way, the behavior of the fluid on a moving domain can be studied.
In this example we are dealing with large displacements, hence a non-linear elasticity model has to be used. In this case the updated Lagrange method, (SP, Section 5.3.2), is used. A Newton method is used to solve the problem in each time-step, but no time-derivatives are used. So we have a quasi stationary problem in each time step.
The deformation of the internal fluid domain is calculated by a pseudo-solid problem. For this problem, the domain is described as a simple, linear solid with prescribed deformation of the (real) solid in the solid part. With the pseudo-solid problem, the deformation of the fluid elements as a result from the deforming solid, can be calculated. Apart from the deformation of the solid, also the solid velocity is prescribed to the fluid as boundary condition. For the fluid, the Arbitrary Lagrangian Eulerian formulation, (ALE), is used. This means that the mesh is updated and the mesh velocity is computed as $\frac{\mathbf{x}^{n+1}-\mathbf{x}^n}{\Delta t}$, where $\mathbf{x}^n$ is the position vector at time t^n. In the convective term, the mesh velocity is subtracted from the Eulerian velocity in order to get the contribution of the mesh deformation. In SEPRAN this contribution is taken into account by setting integer coefficient 15 in the input for Navier-Stokes (See SP, Section 7.1.5).
The fluid problem is solved by Taylor-Hood elements.
The mesh file is given by

```
# domain_def.msh
#
#  mesh file for 2d deformation of solid mesh
#  See Manual Standard Elements Section 5.1.4
#  Author: Martijn Booij 2007
#
#  To run this file use:
#     sepmesh domain_def.msh
#
```

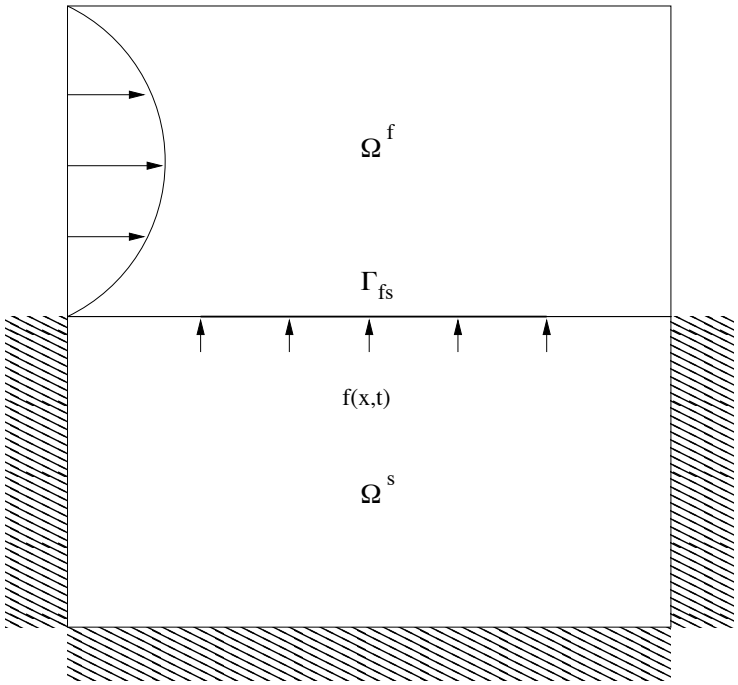


Figure 7.4.2.1: Fluid and solid domain

```
#  Creates the file meshoutput
#
#  Define some general constants

constants

   integers
      n1 = 4               # Number of elements along clamped interface
      n2 = 12              # Number of elements along free part of interface
      m = 10               # Number of elements along y-direction for both
                           # solid and fluid domain

      shape_cur = 2        # Type of elements along curves (quadratic)
      shape_sur1 = 6       # Type of elements in solid domain
                           # (bi-quadratic quadrilaterals)
      shape_sur2 = 4       # Type of elements in fluid domain
                           # (quadratic triangles)
   reals
      height_solid   = 5   # Height of solid domain
      height_fluid   = 10  # Top of fluid domain
      length         = 10  # Length of domain
      length_clamped = 2   # Length of clamped part
      right_clamped = length-length_clamped  # start of right-hand clamped part
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
```

```
#
   points           # See Users Manual Section 2.2
      p1 = (   0,          height_solid) # Left upper point of solid domain
      p2 = (length_clamped,height_solid) # End of left clamped part
      p3 = (right_clamped, height_solid) # Start of left clamped part
      p4 = (  length,      height_solid) # Right upper point of solid domain
      p5 = (  length,      height_fluid) # Right upper point of fluid domain
      p6 = (       0,      height_fluid) # Left upper point of fluid domain
      p7 = (       0,                 0) # Left under point of solid domain
      p8 = (  length,                 0) # Right under point of solid domain
#
#  curves
#
   curves           # See Users Manual Section 2.3

      c1=line shape_cur (p1,p2,nelm=n1)  # left-hand clamped part of interface
      c2=line shape_cur (p2,p3,nelm=n2)  # free part of interface
      c3=line shape_cur (p3,p4,nelm=n1)  # right-hand clamped part of interface
      c4=line shape_cur (p4,p5,nelm=m)   # right-hand side of fluid domain
      c5=translate c7 ( p6, -p5 )        # upper part of fluid domain
      c6=translate c4 ( p1, p6 )         # left-hand side of fluid domain
      c7 = curves(c1, c2, c3)            # interface
      c8 = translate c7 ( p7,-p8 )       # lower part of solid domain
      c9 = line shape_cur (p8,p4,nelm=m) # right-hand side of solid domain
      c10 = translate c9 ( p7, p1 )      # left-hand side of solid domain
      c11 = curves(c1,c3)                # Clamped part of interface
      c12 = curves(c8,c9,c10,c11)        # Clamped part of solid
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4

      s1=rectangle shape_sur1 (c8,c9,-c7,-c10)   # solid domain
      s2=rectangle shape_sur2 (c7,c4,-c5,-c6)    # fluid domain
#
#  Couple surfaces to element groups
#
   meshsurf
      selm1 = ( s1 )      # solid
      selm2 = ( s2 )      # fluid

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2
end
```

Hence in the solid domain we use bi-quadratic quadrilaterals, whereas in the fluid domain quadratic triangles are used.

To do the computation we need a main program in which we define the time-dependent load in a function subroutine FUNCCF and define the initial value for the x-component of the velocity in a function subroutine FUNC. This results in the following main program:

```
      program domain_def
      call sepcom ( 0 )
      end

!     --- Function funccf is used to define the load as function of time
!         See Introduction Manual, Section 5.5.3

      double precision function funccf ( ichoice, x, y,z )
      implicit none

      integer ichoice;
      double precision x, y, z

!     --- Include common ctimen for time t
!         and consta for pi

      include 'SPcommon/ctimen'
      include 'SPcommon/consta'

      if ( ichoice== 1 ) then

!     --- Load function

         funccf = 75d0*(0.5d0-0.5d0*cos((x-2d0)*pi/3d0))*
     +            sin(t)*(x-2d0)/6d0

      end if  ! ( ichoice== 1 )

      end

!     --- Function func to define the u-velocity at start

      double precision function func ( ichoice, x, y, z )
      implicit none
      integer ichoice
      double precision x, y,z

      if ( ichoice==1 ) then

!     --- Parabolic velocity profile

         func = -0.08d0*(y-10d0)*(y-5d0)

      end if  ! ( ichoice==1 )

      end
```

The corresponding input file domain_def.prb uses the following vectors:

u	the displacement vector per time step (solid domain).
un	the global displacement vector, with respect to the original solid domain.
velopres	contains the velocity and pressure in the fluid domain.
meshdisp	the mesh displacement. ll
fluidmeshvelo	the mesh velocity in fluid domain.
coor_old	the coordinate vector $\mathbf{x}^n$ in fluid domain.
coor_new	the coordinate vector $\mathbf{x}^{n+1}$ in fluid domain.

In the program the following steps are performed.

Create the initial vectors for velocity and total displacement.
Copy coordinates in coor_old.
for $t = t_0$ step Δt to $t = t_1$ **do**
  Set **u** equal to 0.
  Set time parameters.
  Solve the non-linear elastic equations to get **u**.
  Compute mesh update by solving the pseudo solid problem with **u** as boundary condition.
  Update mesh.
  un := **un** + **u**
  Copy coordinates in coor_new.
  Compute mesh velocity.
  Compute new velocity and pressure by performing one time step. Use the mesh velocity as boundary condition on the interface.
  coor_old := coor_new.
end for

The corresponding input file is:

```
# domain_def.prb
#
#  problem file for 2d deformation of solid mesh
#  See Manual Standard Elements Section 5.1.4
#  Author: Martijn Booij 2007
#
#  To run this file use:
#     sepcomp domain_def.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      solid = 1     # problem number corresponding to solid
      pseudo = 2    # problem number corresponding to pseudo solid problem
                    # for adapting the mesh
      fluid = 3     # problem number corresponding to fluid
      free  = 2     # Curve number of free part of interface
      clamped = 12  # Curve number of clamped part of solid
      fixed = 11    # Curve number of fixed part of fluid
      wall = 5      # Curve number of wall in fluid
```

```
      inflow = 6    # Curve number of inflow boundary of fluid
      outflow = 4   # Curve number of outflow boundary of fluid
      interface = 7 # Curve number of interface
   reals
      rho = 1        # density
      eta = 1        # viscosity
      eps = 1e-10    # penalty parameter
      t0 = 0         # start time
      t1 = 3.8       # end time
      dt = 0.01      # time step
      dt_print = 0.1 # time step for output

   vector_names
      u              # Displacement vector per pseudo time step
      un             # Total displacement vector
      velopres       # fluid problem solution

      meshdisp       # mesh displacements
      fluidmeshvelo  # fluid mesh velocity
      coor_old       # coordinates before mesh update
      coor_new       # coordinates after mesh update
end
#
#  Define the type of problems to be solved
#  See Users Manual Section 3.2.2
#
problem  solid      # Problem definition corresponding to the solid

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1 = 200              # In the solid domain we use type 200
                                # Non-linear mechanical elements
                                # using the updated Lagrange approach
                                # See Manual Standard Problems Section 5.3.2
      elgrp2 = 0                # No contribution in the fluid domain
   natbouncond                  # Define types of boundary elements
      bngrp1 = 210              # Element for prescribed load corresponding
                                # to type 200
   bounelements                 # Define where the boundary elements are used
      belm1 = curves (c free)   # Prescribed load on free part of interface
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
      curves(c clamped)         # All components are prescribed at clamped
                                # part of boundary

problem   pseudo      # Problem definition corresponding to the pseudo solid
                      # problem, which is used to update the mesh

   types                        # Define types of elements,
      elgrp1 = 250              # Linear elasticity problem in solid problem
      elgrp2 = 250              # Linear elasticity problem in fluid problem
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
      surfaces(s1)              # The displacement in the solid is given
```

```
      curves(c outflow)        # No displacement at fluid outflow
      curves(c inflow)         # No displacement at fluid inflow
      curves(c wall)           # No displacement at fluid wall

problem   fluid       # Problem definition corresponding to the  fluid
   types                       # Define types of elements,
      elgrp1 = 0               # No fluid in solid domain
      elgrp2 = 903             # Taylor-Hood elements in fluid domain
                               # Navier-Stokes, integrated approach
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
      degfd1, degfd2 = curves(c interface) # Given velocity at interface
      degfd2 = curves(c outflow)           # Tangential velocity at outflow is 0
      degfd1, degfd2 = curves(c wall)      # Zero velocity at wall
      degfd1, degfd2 = curves(c inflow)    # Given velocity at inflow

   renumber levels (1,2),(3)   # Renumbering of unknowns is necessary to
                               # avoid zero pivots
                               # This is characteristic for Navier-Stokes
end

#  Define the structure of the large matrices
#  See Users Manual Section 3.2.4
#  The solid and fluid are solved by iterative solvers, hence a
#  compact storage is used. The matrices are unsymmetric
#  The pseudo solid problem is symmetric and solved by a direct solver
#  Hence storage scheme is profile

matrix
   storage_scheme = compact, problem solid
   storage_scheme = profile, symmetric, problem pseudo
   storage_scheme = compact, problem fluid
end

#  Definition of essential boundary conditions
#  See Users Manual Section 3.2.5
#  Solid problem

essential boundary conditions, sequence_number =  solid, problem =  solid
#       zero bc, hence no input required
end

#  Pseudo solid problem for mesh update

essential boundary conditions, sequence_number =  pseudo , problem =  pseudo
   surfaces(s1) = vector = u  ! The displacement in the solid part is given by u
end

# Fluid problem
# At the free surface we use the fluid mesh velocity as stored in
# fluidmeshvelo
# At inflow the x-component is a quadratic function with maximum value 0.5

essential boundary conditions, sequence_number =  fluid , problem =  fluid
```

```
   curves(c free), degfd1, degfd2,vector = fluidmeshvelo
   curves(c inflow), degfd1, quadratic, max= 0.5

end

#  Define the coefficients for the solid problem
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 5.3.2

coefficients, sequence_number =  solid , problem =  solid
# internal elements
   elgrp1 (nparm = 45)  #  Type 200
      icoef2 = 0     # stress strain relation 0 = 2d plain strain
      icoef4 = 1     # 1 compressible elastic solid
      coef10 = 50    # shear modulus
      coef11 = 40    # bulk modulus

# boundary elements
   bngrp1 (nparm=15) #  Type 210
      coef7=func=1   # force in global y-direction
end

#  Define the coefficients for the pseudo solid problem
#  to update the mesh

coefficients, sequence_number =  pseudo , problem =  pseudo
# fluid
   elgrp1, (nparm = 45)
      coef6 =  10    # Young's modulus
      coef7 =  0.3   # Poisson ratio;
# structure
   elgrp2, (nparm = 45)
      coef6 =     10      # Young's modulus
      coef7 =     0.3     # Poisson ratio
end

#  Define the coefficients for the fluid problem

coefficients, sequence_number =  fluid , problem =  fluid
   elgrp2 ( nparm=20 )         # The coefficients are defined by 20 parameters
      icoef2 = 1          # 2:  type of constitutive equation (1=Newton)
      icoef3 = 3          # 2:  numerical integration rule (Gauss)
      icoef5 = 1          # 5:  Type of linearization (1=Picard)
      coef6  =  eps       # 6:  Penalty parameter
      coef7  =  rho       # 7:  Density
      coef12 =  eta       #12:  Value of eta (viscosity)
      icoef15  = fluidmeshvelo    # mesh velocity sequence number
end

# Define the structure of the program
# See Users Manual Section 3.2.3

structure

  # Create start vectors for total displacement in solid domain un and
```

```
# velocity and pressure in fluid domain velopres

 create_vector, un, sequence_number = 1         # un = 0
 create_vector, velopres, sequence_number = 2   # v_1 = parabolic, v_2 = 0

# write start vectors to file sepcomp.out

 output

# Store coordinates in vector coor_old (only for fluid problem)

 copy_coor coor_old, problem = fluid

# Solve time dependent problem

 start_time_loop

   # Create initial value for displacement vector per pseudo time step
    create_vector, sequence_number = 1,  u   # u = 0, solid domain only

   # Raise actual time, set time step and so on
   # No action is performed

    time_integration, sequence_number =  solid

    ### Solve system of non-linear equations to get new increment vector
    solve_nonlinear_system, vector = u

    print_time

   # Compute the mesh displacement by solving the pseudo solid problem
   # Boundary conditions are given by the displacement u

    prescribe_boundary_conditions, sequence_number =  pseudo,  meshdisp

    solve_linear_system seq_coef =  pseudo,  meshdisp

   # Use the computed mesh displacement to deform the mesh

    deform_mesh,  meshdisp
    plot_mesh

   # The total displacement is the sum of the original total displacement
   # and the displacement vector per pseudo time step

    un = un + u
    plot_vector un, factor = 1

  #   Because the ALE formulation for the fluid is used, the velocity of
  #   the fluid domain nodes is computed from co-ordinates of present
  #   and previous co-ordinates
  #   We copy the present coordinates into coor_new, which is defined
  #   over the fluid domain only
  #   Next the mesh velocity in fluid domain is computed by
  #   (coor_new-coor_old)/dt
```

```
    #   And finally the new coorinates are copied into coor_old

      copy_coor, coor_new, problem = fluid
      fluidmeshvelo = mesh_velocity(coor_new,coor_old)
      coor_old = coor_new

     # Compute the velocity by solving one step of the fluid problem
     # Use the computed mesh velocity as boundary condition on the free
     # part of the interface

      time_integration, sequence_number =  fluid,  velopres
      plot_vector velopres, factor = 0.3
      plot_coloured_levels velopres, degfd 3

     # Write all vectors to sepcomp.out

      output

   end_time_loop

end

#  Define the time integration process
#  In this case it is done in two separate input blocks
#  See Users Manual Section 3.2.15

#  First input block defines the initial time, time step and end time
#  Both for computing and output
#  It is also used to raise the actual time, but not to perform any action
#  during the time step

time_integration, sequence_number  =  solid
   method = stationary  # The solid problem is stationary
   tinit = t0           # initial time
   tend =  t1           # end time
   tstep =  dt          # time step
   print_level = 1

   toutinit = t0        # initial time for output
   toutend  =  t1       # end time for output
   toutstep =  dt_print # time step for output
end

#  Time integration of fluid equation (Navier-Stokes)
time_integration, sequence_number =  fluid
   method   = theta                  # theta method
   theta = 1                         # theta = 1, hence Euler implicit
   reuse_time_parameters             # The time parameters have been set
                                     # in the first time integration block
   non_linear_iteration              # Perform a non linear iteration for
                                     # each time step
   print_level = 2                   # Produce extra output
   max_iter = 5                      # Maximum number of iterations
   seq_coefficients        =  fluid
   seq_boundary_conditions =  fluid
```

```
   seq_solution_method     =  fluid
end

# Definition of iteration for non linear equations (solid problem)
#  See Users Manual Section 3.2.9

nonlinear_equations, sequence_number = solid
   number_of_couple_equations = 1   # necessary, since only one non-linear
                                    # problem must be solved
   global_options, maxiter = 50, miniter = 1, accuracy = 1d-4//
      criterion = relative,  print_level = 1,  at_error= return //
      iteration_method = newton
   equations 1
     fill_coefficients = solid
end

#  Create displacement vector and set equal to 0
#  See Users Manual Section 3.2.10
#  No extra input required

create vector, sequence_number = 1 , problem =  solid
end

#  Initialize fluid problem solution vector
#  The v-velocity is 0, the u veclocity is a quadrtic function of y
#  Defined by a function subroutine func

create vector, sequence_number = 2   , problem =  fluid
   degfd1, func = 1
end

#  Input for the linear solver
#  Only for the fluid problem
#  See Users Manual Section 3.2.8

solve, sequence_number =  fluid
   iteration_method = bicgstab, preconditioning = ilu, accuracy =  eps,//
   print_level = 0, termination_crit = rel_residual, start = old_solution
end
```

Figures (7.4.2.2)-(7.4.2.5), show the mesh at $t = 0.1$, and $t = 2, 3, 4$ respectively.

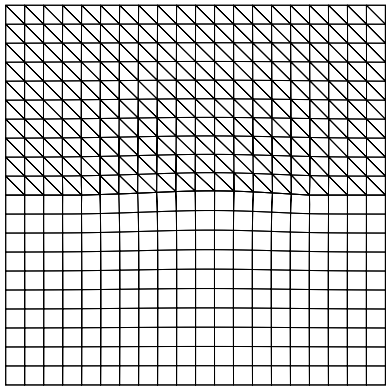


Figure 7.4.2.2: Mesh at t=0.1

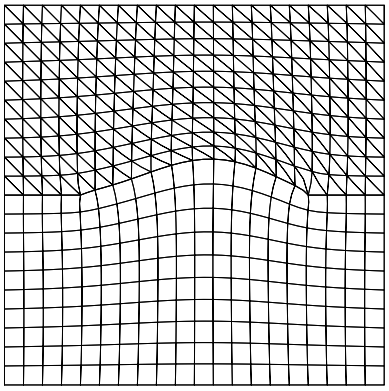


Figure 7.4.2.3: Mesh at t=1

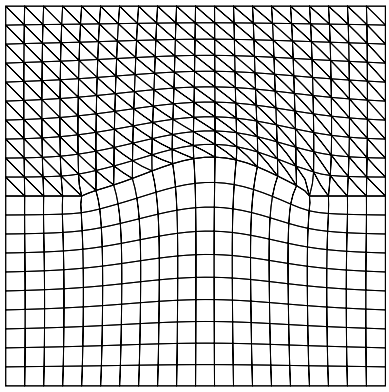


Figure 7.4.2.4: Mesh at t=2

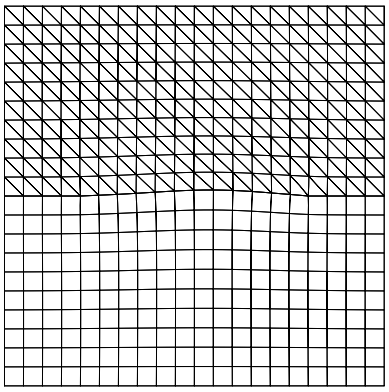


Figure 7.4.2.5: Mesh at t=3

Figures (7.4.2.6)-(7.4.2.9), show the displacement of the solid nodes at $t = 0.1$, and $t = 2, 3, 4$ respectively.

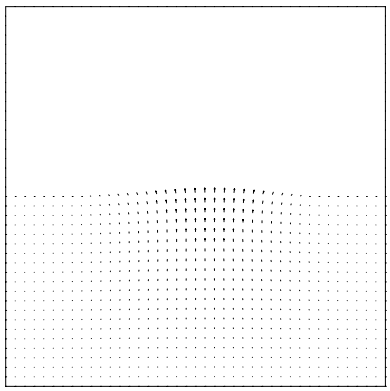


Figure 7.4.2.6: Displacement of solid nodes at t=0.1

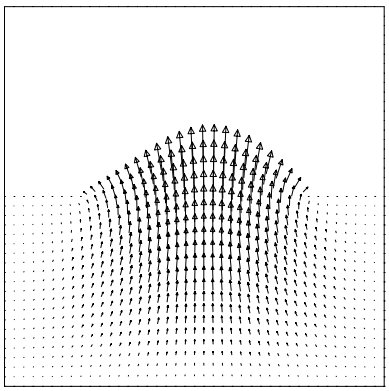


Figure 7.4.2.7: Displacement of solid nodes at t=1

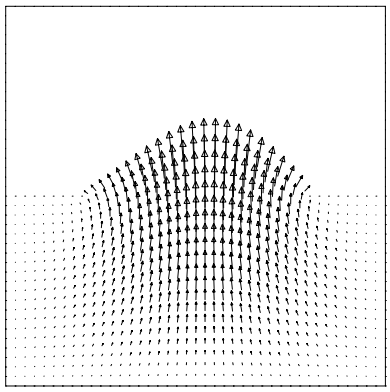


Figure 7.4.2.8: Displacement of solid nodes at t=2

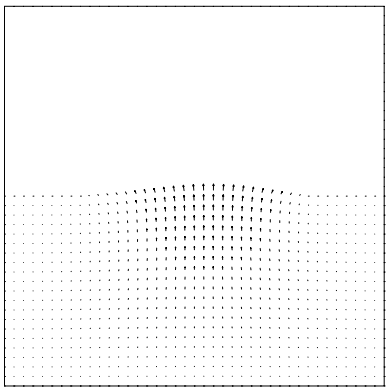


Figure 7.4.2.9: Displacement of solid nodes at t=3

Figures (7.4.2.10)-(7.4.2.13), show the fluid velocity at $t = 0.1$, and $t = 2, 3, 4$ respectively.

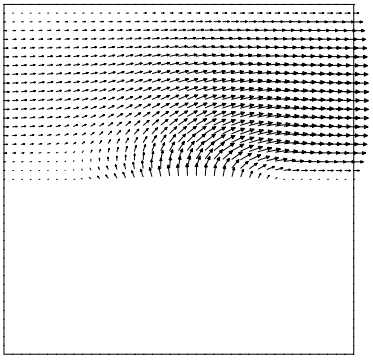


Figure 7.4.2.10: Fluid velocity at t=0.1

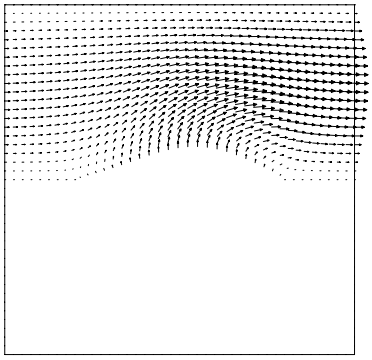


Figure 7.4.2.11: Fluid velocity at t=1

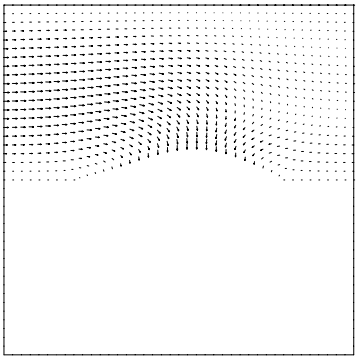


Figure 7.4.2.12: Fluid velocity at t=2

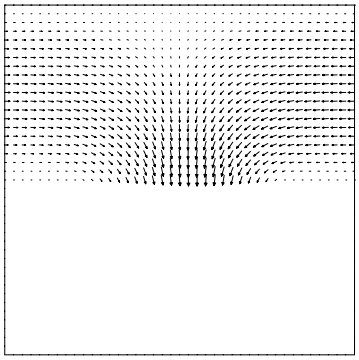


Figure 7.4.2.13: Fluid velocity at t=3

Figures (7.4.2.14)-(7.4.2.17), show the pressure levels at $t = 0.1$, and $t = 2, 3, 4$ respectively.

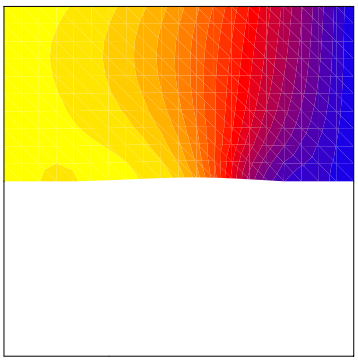


Figure 7.4.2.14: Pressure level at t=0.1

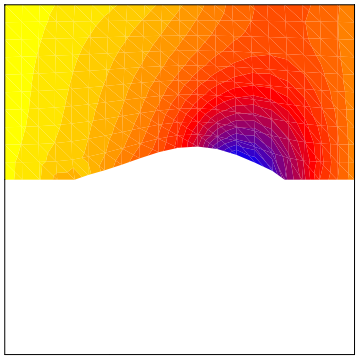


Figure 7.4.2.15: Pressure level at t=1

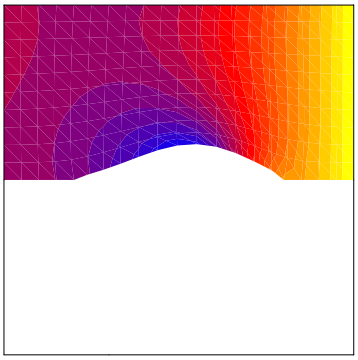


Figure 7.4.2.16: Pressure level at t=2

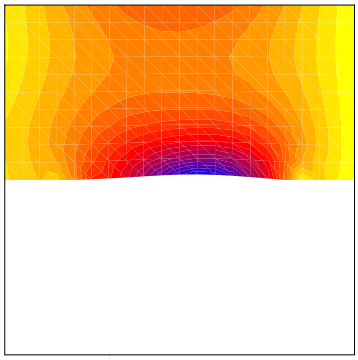


Figure 7.4.2.17: Pressure level at t=3

In the post processing phase the displacement of the node at (0.5,0.5) is plotted.

```
# domain_def.pst
#
#  post processing file for 2d deformation of solid mesh
#  See Manual Standard Elements Section 5.1.4
#  Author: Martijn Booij 2007
#
#  To run this file use:
#     seppost domain_def.pst
#
#  Reads the file meshoutput and sepcomp.out
#
postprocessing
   time history plot point ( 5, 5 ) un, degfd = 2
end
```

Figure 7.4.2.18 shows the computed displacement of this point.

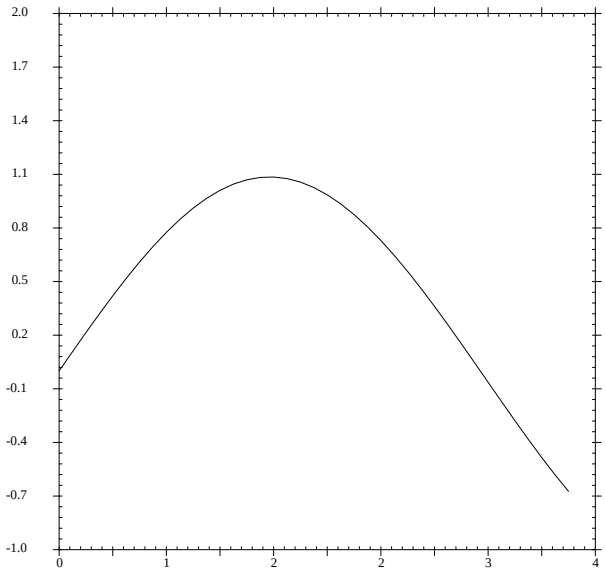


Figure 7.4.2.18: displace of node (0.5,0.5) in time

7.5 Methods to compute fluid flow in the presence of an obstacle

7.5.1 A simple stationary obstacle in a two-dimensional fluid

In this section we show the various methods treated in Section 7.5 to compute the flow around a stationary obstacle.
To get these examples into your local directory use:

```
sepgetex obstaclexx_y
```

with xx a two-digit number and y a one-digit number.
and to run it use:

```
sepmesh obstaclexx_y.msh
sepcomp obstaclexx_y.prb
seppost obstaclexx_y.pst
```

After the first and last step you may view the results using sepview.

The following values for `xx` are available:

```
xx = 01, 02, 03
```

and for `y`:

```
y = 1 to 4
```

Not all combinations of `xx` and `y` have been programmed yet.
`xx` has the following meaning:

01 The mesh is adapted to the obstacle. In this case the boundary of the obstacle is also boundary of the fluid domain. Section 7.5.1.1.

02 A fixed mesh for the fluid is used independent on the obstacle. The velocities in all nodes of the fluid mesh that are inside or on the obstacle are set to 0. This is the most primitive approach. In fact the computational obstacle is smaller than the actual one. Section 7.5.1.2.

03 A fixed mesh for the fluid is used independent on the obstacle. The velocities in all nodes of the fluid mesh that are inside or on the obstacle are set to 0, like in the case 02.
All intersections of the boundary of the obstacle and the fixed fluid mesh, are computed and if an intersection point is not a nodal point of the fluid mesh we add the constraint that the velocity in that point must be zero by use of Lagrangian multipliers. This is a kind of fictitious domain approach. Section 7.5.1.3.

04 A fixed mesh for the fluid is used independent on the obstacle. The mesh is adapted by computation of intersections with the obstacle. Elements that are intersected are subdivided provided the intersection is not too close to the nodes. (approximately adapted grid method). Elements in the new mesh are considered to be either inside or outside the obstacle, hence the obstacle is in general approximated by another shape, which is close to the original one. Section 7.5.1.4.

`y` has the following meaning:

1 The Taylor-Hood linear triangle (mini element) is used.

2 The Taylor-Hood quadratic triangle is used.

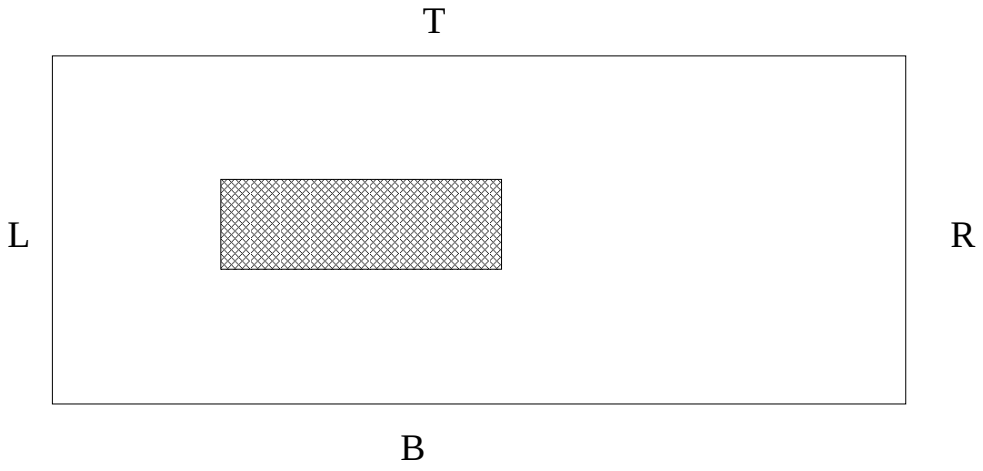


Figure 7.5.1.1: Sketch of channel with rectangular obstacle

3 The Crouzeix-Raviart bilinear quadrilateral is used.

4 The Crouzeix-Raviart quadratic triangle is used.

The problem that we consider is that of a very simple rectangular obstacle in a channel as sketched in Figure 7.5.1.1

The obstacle is not moving, hence the velocity at the boundary of the solid is zero.
At the left-hand side we have a quadratic velocity profile with maximum velocity 1. Top and bottom of the channel are no-slip walls and at the right-hand side we have outflow.
The density ρ is chosen equal to 1 and the viscosity μ equal to 0.01.

Boundary conditions in this case are:

inflow:	v_1 quadratic, $v_2 = 0$ (given velocity)
outflow:	$\sigma^{nn} = 0$, $\sigma^{nt} = 0$ (stress free)
walls:	$\mathbf{v} = \mathbf{0}$ (noslip)
obstacle:	$\mathbf{v} = \mathbf{0}$ (noslip)

In the following subsections we show the input and results of the various methods.

7.5.1.1 Mesh adapted to the obstacle

In this case only the fluid region is covered with elements.
This method is of course the most accurate and may be used as a reference for all *fixed* mesh methods.

The mesh is defined by the following mesh input file

```
# obstacle01_1.msh
#
#  Mesh for 2d obstacle domain example
#  The problem considered here is that of a fixed obstacle in a fluid
#
#  In this example we compute the flow around the obstacle by adapting the
#  mesh to the obstacle
#  The solution can be used as reference for the other methods treated
#  in this Section
#
#  In this specific example we use Taylor-Hood linear triangles (mini element)
#  See Manual Examples Section 7.5.1
#
#  To run this file use:
#     sepmesh obstacle01_1.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      # Fluid region
      x_left = 0              # Left-hand side x-coordinate of fluid domain
      x_right = 4             # Right-hand side x-coordinate of fluid domain
      y_low = 0               # Lower y-coordinate of fluid domain
      y_top = 1               # Upper y-coordinate of fluid domain
      # obstacle
      x_left_obs = 1          # Left-hand side x-coordinate of obstacle
      x_right_obs = 2         # Right-hand side x-coordinate of obstacle
                              # at lower boundary
      y_low_obs = 0.25        # Lower y-coordinate of obstacle
      y_top_obs = 0.55        # Upper y-coordinate of obstacle
      unit_length = 0.075     # Unit_length for coarseness
      coarse_obst = 0.5       # Relative length for obstacle
   integers
      lin = 1                 # Type of elements along lines (linear)
      surf = 3                # Type of elements in surface (linear triangles)
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points, See Users Manual Section 2.2
#
   coarse ( unit =  unit_length )      # defines the length of the elements
#
```

```
#  user points
#
   points           # See Users Manual Section 2.2


      # Fluid mesh
         p1 = ( x_left,  y_low)       # Left under point
         p2 = ( x_right,  y_low)      # Right under point
         p3 = ( x_right,  y_top)      # Right upper point
         p4 = ( x_left,  y_top)       # Left upper point
      # Obstacle
         p11 = ( x_left_obs,  y_low_obs, coarse_obst)   # Left under point
         p12 = ( x_right_obs,  y_low_obs, coarse_obst)  # Right under point
         p13 = ( x_right_obs,  y_top_obs, coarse_obst)  # Right upper point
         p14 = ( x_left_obs,  y_top_obs, coarse_obst)   # Left upper point
      # Extra point
         p20 = ( x_left,  y_low_obs)   # extra point at left hand side
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    #
      # Fluid mesh
         c1 = cline  lin ( p1, p2 )      # lower wall
         c2 = cline  lin ( p2, p3 )      # outflow boundary
         c3 = cline  lin ( p3, p4 )      # upper wall
         c4 = curves(c5,c6)              # inflow boundary
         c5 = cline  lin ( p4, p20 )     # upper part inflow boundary
         c6 = cline  lin ( p20, p1 )     # lower part inflow boundary
      # Obstacle
         c11 = cline  lin ( p11, p12 )   # lower part
         c12 = cline  lin ( p12, p13 )   # right-hand side
         c13 = cline  lin ( p13, p14 )   # upper part
         c14 = cline  lin ( p14, p11 )   # left-hand side
         c20 = curves(c11,c12,c13,c14)   # Complete obstacle
      # Line from extra point to obstacle
         c10 = cline  lin ( p20, p11 )
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      # Fluid mesh
         s1 = general  surf(c1,c2,c3,c5,c10,-c20,-c10,c6)
   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2
end
```

Figure 7.5.1.2 shows the curve numbers used in this example and Figure 7.5.1.3 the corresponding mesh.

The input file for sepcomp is more or less standard.
We show the one used for the mini element.

```
# obstacle01_1.prb
#
#  Problem file for 2d obstacle domain example
```

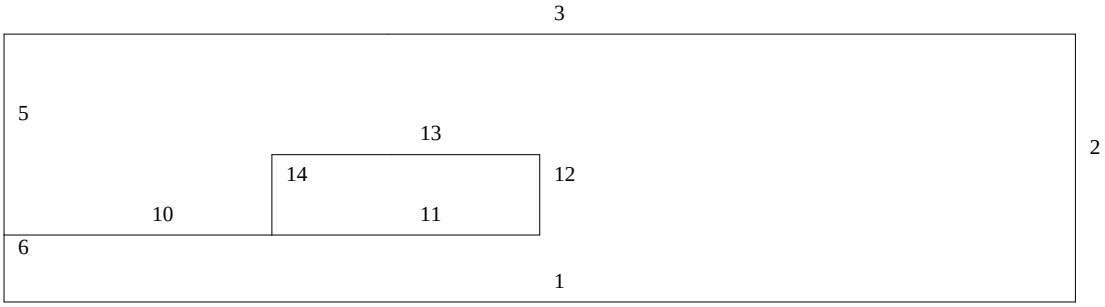


Figure 7.5.1.2: Curves for the obstacle in the fluid

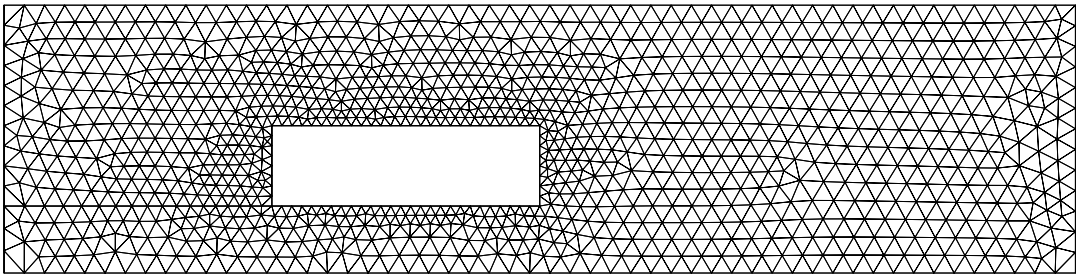


Figure 7.5.1.3: Mesh for the obstacle in the fluid

```
#  The problem considered here is that of a fixed obstacle in a fluid
#
#  In this example we compute the flow around the obstacle by adapting the
#  mesh to the obstacle
#  The solution can be used as reference for the other methods treated
#  in this Section
#
#  In this specific example we use Taylor-Hood linear triangles (mini element)
#  See Manual Examples Section 7.5.1
#
#  To run this file use:
#     sepcomp obstacle01_1.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
```

```
set warn off    ! suppress warnings
constants           # See Users Manual Section 1.4
   reals
      rho       = 1         # density of fluid
      eta       = 0.01      # dynamic viscosity
   vector_names
      velocity_pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
     elgrp1=903                # Type number for Navier-Stokes
                               # Taylor-Hood element

   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2

     degfd1, degfd2 = curves(c1)            # no-slip bottom wall
     degfd1, degfd2 = curves(c3)            # no-slip top wall
     degfd1, degfd2 = curves(c4)            # inlet
     degfd1, degfd2 = curves(c20)           # obstacle no-slip

end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions
   curves(c4), degfd1, quadratic, max=1   # The u-component of the velocity at
                                          # inflow is parabolic
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=2, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Sections 7.1

coefficients, problem=1
```

```
   elgrp1 ( nparm=20 )     # The coeffs are defined by 20 parameters
      icoef2 = 1           # type of constitutive equation (1=Newton)
      icoef5 = 0           # Type of linearization (0=Stokes flow)
      coef7  =  rho        # Density
      coef12 =  eta        # Value of eta (dynamic viscosity)

end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is part is not necessary
#

structure                   # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions, velocity_pressure
   solve_nonlinear_system, velocity_pressure
 # Write the results to a file
   output
end
```

Post processing can be performed by the following

```
#  obstacle01_1.pst
#  Input file for postprocessing for 2d obstacle domain example
#  The problem considered here is that of a fixed obstacle in a fluid
#
#  In this example we compute the flow around the obstacle by adapting the
#  mesh to the obstacle
#  The solution can be used as reference for the other methods treated
#  in this Section
#
#  In this specific example we use Taylor-Hood linear triangles (mini element)
#  See Manual Examples Section 7.5.1
#
#
#  To run this file use:
#     seppost obstacle01_1.pst > obstacle01_1.out
#
#  Reads the files meshoutput and sepcomp.out
#
```

```
postprocessing                    # See Users Manual Section 5.2

#  Plot the mesh

   plot mesh

#
#  compute the stream function
#  See Users Manual Section 5.2
#  store in stream_function

   compute stream function velocity_pressure

#  Plot the results
#  See Users Manual Section 5.4

   plot vector velocity_pressure              # Vector plot of velocity
   plot contour velocity_pressure, degfd=3    # Contour plot of pressure
   plot coloured contour velocity_pressure, degfd=3
   plot contour stream_function      # Contour plot of stream function
   plot coloured contour stream_function

end
```

Figure 7.5.1.4 shows the coloured isobars and Figure 7.5.1.5 the coloured stream function levels.

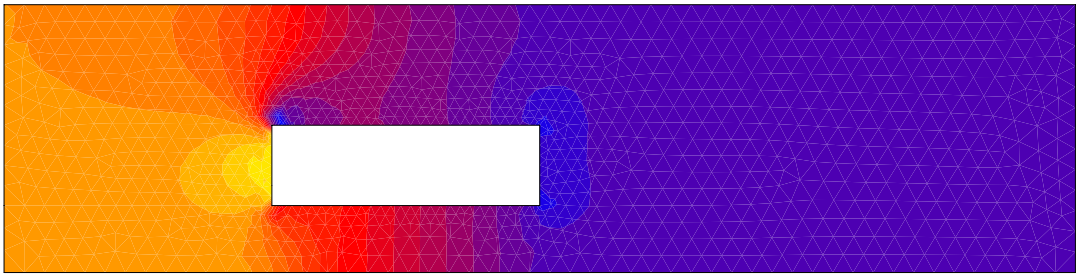


Figure 7.5.1.4: Coloured isobars

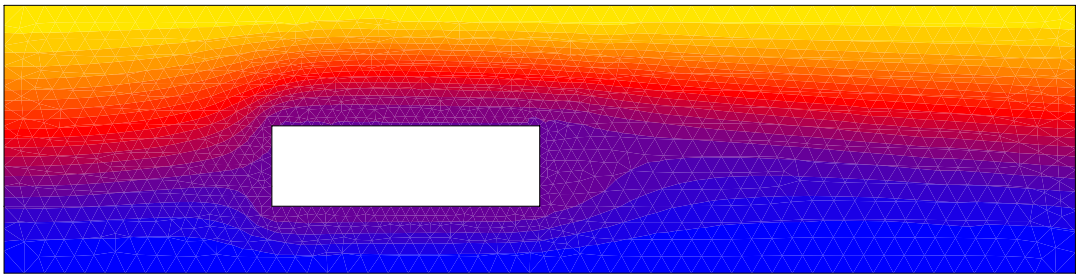


Figure 7.5.1.5: Stream function levels

7.5.1.2 Fixed mesh, primitive approach

In the primitive approach the start is a fixed mesh for the fluid, that is not adapted to the obstacle. So in fact we have the same mesh as if we did not have an obstacle. Next all elements that are completely inside the obstacle are marked and all nodes inside these elements get the velocity of the obstacle (in this case 0). So what happens is, is that the obstacle is shrinked to the set of elements in the fluid mesh, that are completely inside the obstacle. The obstacle gets smaller and hence also its effect. This approach is very easy to apply, and if the mesh is sufficiently fine near the boundary of the obstacle also not too inaccurate. In this case the mesh is much simpler than in Section 7.5.1.1 Extra is the introduction of an obstacle in the mesh. The mesh is defined by the following mesh input fileex-chap-7.5.1.1.

```
# obstacle02_1.msh
#
#  Mesh for 2d obstacle domain example
#  The problem considered here is that of a fixed obstacle in a fluid
#
#  In this example we compute the flow around the obstacle with a fixed mesh
#  The obstacle is defined as an obstacle and the internal velocities are
#  set to 0 (primitive approach)
#
#  In this specific example we use Taylor-Hood linear triangles (mini element)
#  See Manual Examples Section 7.5.1
#
#  To run this file use:
#     sepmesh obstacle02_1.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      # Fluid region
      x_left = 0               # Left-hand side x-coordinate of fluid domain
      x_right = 4              # Right-hand side x-coordinate of fluid domain
      y_low = 0                # Lower y-coordinate of fluid domain
      y_top = 1                # Upper y-coordinate of fluid domain
      # obstacle
      x_left_obs = 1           # Left-hand side x-coordinate of obstacle
      x_right_obs = 2          # Right-hand side x-coordinate of obstacle
                               # at lower boundary
      y_low_obs = 0.25         # Lower y-coordinate of obstacle
      y_top_obs = 0.55         # Upper y-coordinate of obstacle
      unit_length = 0.075      # Unit_length for coarseness
   integers
      lin = 1                  # Type of elements along lines (linear)
      surf = 3                 # Type of elements in surface (linear triangles)
end
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points, See Users Manual Section 2.2
#
```

```
   coarse ( unit =  unit_length )      # defines the length of the elements
#
#  user points
#
   points           # See Users Manual Section 2.2


      # Fluid mesh
         p1 = ( x_left,  y_low)       # Left under point
         p2 = ( x_right,  y_low)      # Right under point
         p3 = ( x_right,  y_top)      # Right upper point
         p4 = ( x_left,  y_top)       # Left upper point
      # Obstacle
         p11 = ( x_left_obs,  y_low_obs)   # Left under point
         p12 = ( x_right_obs,  y_low_obs)  # Right under point
         p13 = ( x_right_obs,  y_top_obs)  # Right upper point
         p14 = ( x_left_obs,  y_top_obs)   # Left upper point
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    #
      # Fluid mesh
         c1 = cline  lin ( p1, p2 )      # lower wall
         c2 = cline  lin ( p2, p3 )      # outflow boundary
         c3 = translate c1 ( p4, p3 )    # upper wall
         c4 = translate c2 ( p1, p4 )    # inflow boundary
      # Obstacle
         c11 = line ( p11, p12, nelm=1 )   # lower part
         c12 = line ( p12, p13, nelm=1 )   # right-hand side
         c13 = line ( p13, p14, nelm=1 )   # upper part
         c14 = line ( p14, p11, nelm=1 )   # left-hand side
         c20 = curves(c11,c12,c13,c14)
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      # Fluid mesh
         s1 = rectangle  surf(c1,c2,-c3,-c4)
#
#  obstacles
#
   obstacles         # See Users Manual Section 2.1
      cobs1 = c20
   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2
end
```

The problem file has the following shape

```
# obstacle02_1.prb
#
#  Problem file for 2d obstacle domain example
#  The problem considered here is that of a fixed obstacle in a fluid
#
```

```
#  In this example we compute the flow around the obstacle with a fixed mesh
#  The obstacle is defined as an obstacle and the internal velocities are
#  set to 0 (primitive approach)
#
#  In this specific example we use Taylor-Hood linear triangles (mini element)
#  See Manual Examples Section 7.5.1
#
#  To run this file use:
#     sepcomp obstacle02_1.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
set warn off    ! suppress warnings
constants           # See Users Manual Section 1.4
   reals
      rho       = 1         # density of fluid
      eta       = 0.01      # dynamic viscosity
   vector_names
      velocity_pressure
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
     elgrp1=903                 # Type number for Navier-Stokes
                                # Taylor-Hood element

   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2

      degfd1, degfd2 = curves(c1)           # no-slip bottom wall
      degfd1, degfd2 = curves(c3)           # no-slip top wall
      degfd1, degfd2 = curves(c4)           # inlet
      degfd1, degfd2 = in_all_obstacle 1    # all velocities in the obstacle
                                            # are set to zero
      degfd3 = in_inner_obstacle 1          # all pressures corresponding to
                                            # elements that are completely
                                            # within the obstacle are set to
                                            # zero, because these elements are skipped

   skip_elements                # skip some elements in order to avoid singular
                                # matrices, due to a pressure not coupled to
                                # free velocities
      inner_obstacle 1          # All elements that are completely within the
                                # obstacle are skipped
                                # Mark that it is not allowed to set the pressure
```

```
                              # inside the obstacle to zero, since that is
                              # not correct
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions
   curves(c4), degfd1, quadratic, max=1   # The u-component of the velocity at
                                          # inflow is parabolic
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=2, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Sections 7.1

coefficients, problem=1

   elgrp1 ( nparm=20 )     # The coeffs are defined by 20 parameters
      icoef2 = 1           # type of constitutive equation (1=Newton)
      icoef5 = 0           # Type of linearization (0=Stokes flow)
      coef7  =  rho        # Density
      coef12 =  eta        #12:  Value of eta (dynamic viscosity)

end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is part is not necessary
```

```
#
structure                    # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions, velocity_pressure
   solve_nonlinear_system, velocity_pressure
 # Write the results to a file
   output
end
```

Figure 7.5.1.6 shows the mesh and Figure 7.5.1.7 the velocity vectors.

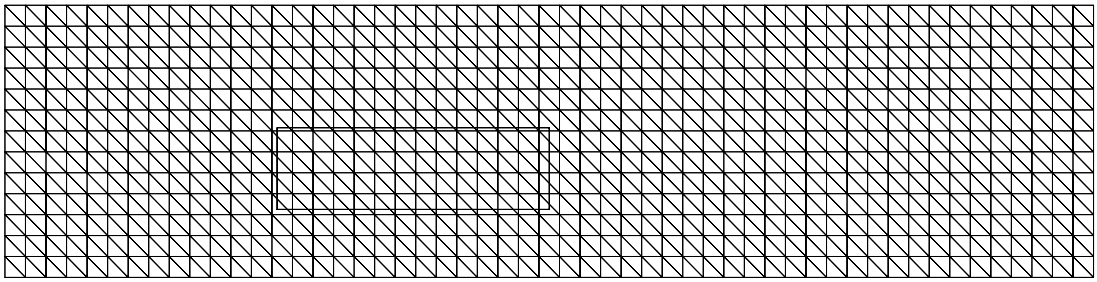


Figure 7.5.1.6: Fixed mesh

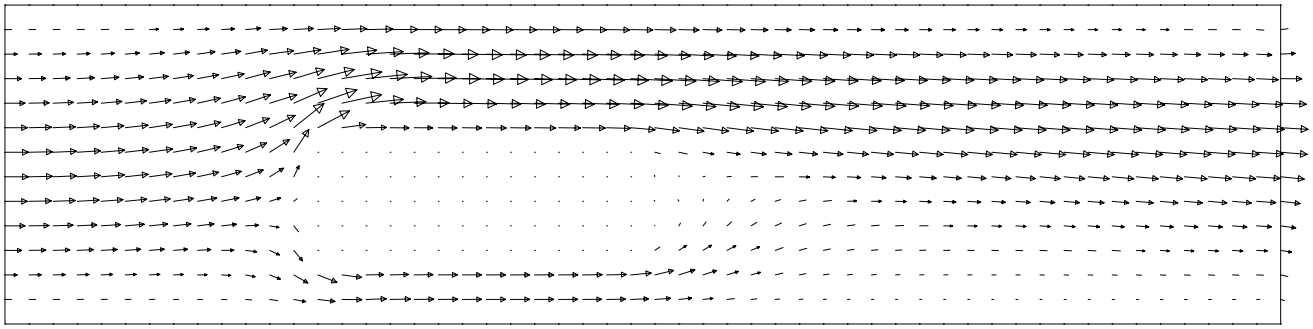


Figure 7.5.1.7: Velocity vectors

7.5.1.3 Fixed mesh, fictitious domain approach

The start of this approach is exactly the same as for the primitive approach. So we use a fixed fluid mesh and the velocities for elements completely inside the obstacle are prescribed (0). However, in order to make the obstacle larger, we consider all intersections of the obstacle with the fluid mesh. Actually the velocities in the intersection points should also be zero, but this is only the case if such an intersection points coincides with a nodal point. Now, however, we demand that the velocity in that point is zero, provided the intersection point is not very close to a node in the obstacle. This requirement is prescribed by means of a constraint and to satisfy this constraint it is necessary to introduce Lagrangian multipliers. This is precisely the fictitious domain approach.
Of course we must avoid that these constraints are linearly dependent, so the Lagrangian multiplier is defined on a limited number of edge elements, in such a way that a singular matrix is avoided.
Note that of we are dealing with linear elements, setting the velocity 0 in one node of an edge and requiring that it is zero in an intermediate point, means actually that the velocity is zero along the whole edge. So in this case the obstacle becomes too wide, and to prescribe as less as possible velocities it is advised to use `exclude_type = 1`, in the creation of the "edge"-elements for the Lagrangian multipliers.

The mesh for this example is completely identical to the mesh in Section 7.5.1.2.

The problem file is given by

```
# obstacle03_1.prb
#
#  Problem file for 2d obstacle domain example
#  The problem considered here is that of a fixed obstacle in a fluid
#
#  In this example we compute the flow around the obstacle with a fixed mesh
#  The obstacle is defined as an obstacle and the internal velocities are
#  set to 0
#  Furthermore the velocity on the obstacle is set to zero using boundary
#  elements of type 922 and the option cross_section_obstacle
#  In this way the velocity condition on the boundary is treated as a constraint
#
#  In this specific example we use Taylor-Hood linear triangles (mini element)
#  See Manual Examples Section 7.5.1
#
#  To run this file use:
#     sepcomp obstacle03_1.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
set warn off    ! suppress warnings
constants           # See Users Manual Section 1.4
   reals
      rho       = 1         # density of fluid
      eta       = 0.01      # dynamic viscosity
      u_obst    = 0         # u-velocity of obstacle
      v_obst    = 0         # v-velocity of obstacle
   vector_names
      velocity_pressure
```

```
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
     elgrp1=903                # Type number for Navier-Stokes
                               # Taylor-Hood element

   natbouncond              # Natural boundary conditions
                            # In this case the natural boundary conditions
                            # are defined in order to introduce intersection
                            # elements between obstacle and fluid mesh
      bngrp1 = 922          # Special type meant for the intersection element
                            # defines constraints and lagrangian multipliers
   bounelements             # Corresponding boundary elements
      belm1 = cross_section_obstacle 1, exclude_type = 1
                            # Boundary elements are defined for the
                            # cross-section
                            # exclude_type defines which elements are excluded
                            # 1: Each point in the flow (outside the obstacle)
                            #    may be connected to only one
                            #    cross-section element
                            # 2: Each point in the region
                            #    (including the obstacle)may be connected to
                            #    only one cross-section element
                            # Default: 1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2

      degfd1, degfd2 = curves(c1)           # no-slip bottom wall
      degfd1, degfd2 = curves(c3)           # no-slip top wall
      degfd1, degfd2 = curves(c4)           # inlet
                       in_inner_obstacle 1  # For points that are only in
                                            # elements that are completely
                                            # in the obstacle both pressure and
                                            # velocity are given
      degfd1, degfd2 = in_boun_obstacle 1   # For the other points in the
                                            # obstacle only the velocity is
                                            # prescribed
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions
   curves(c4), degfd1, quadratic, max=1   # The u-component of the velocity at
                                          # inflow is parabolic
end

# input for non-linear solver
```

```
# See Users Manual Section 3.2.9

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=2, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Sections 7.1

coefficients, problem=1

   elgrp1 ( nparm=20 )     # The coeffs are defined by 20 parameters
      icoef2 = 1           # type of constitutive equation (1=Newton)
      icoef5 = 0           # Type of linearization (0=Stokes flow)
      coef7  =  rho        # Density
      coef12 =  eta        #12:  Value of eta (dynamic viscosity)

   bngrp1 ( nparm=10 )     # The coeffs are defined by 10 parameters
                           # Elements of type 922 require the velocity of the
                           # obstacle as input (coefficients 6 and 7)
      coef6  =  u_obst     # u
      coef7  =  v_obst     # v

end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is part is not necessary
#
structure                  # See Users Manual Section 3.2.3
 # Compute the velocity
   prescribe_boundary_conditions, velocity_pressure
   solve_nonlinear_system, velocity_pressure
 # Write the results to a file
   output
end
```

Figure 7.5.1.8 shows the velocity vectors in this case.

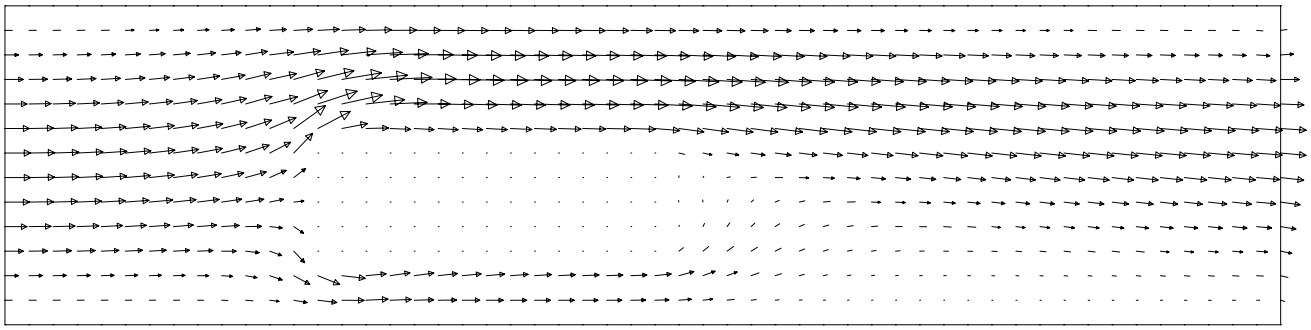


Figure 7.5.1.8: Velocity vectors

7.5.1.4 Fixed mesh, approximated adapted mesh approach

The start of this approach is same fixed fluid mesh as in Section 7.5.1.3. The same intersections are created. However, in this case elements that are intersected by the obstacle, are subdivided into subelements, provided the intersection is not too close to a node of the fixed mesh. If an intersection point is within a distance ϵ times the edge length of a node, this node is considered to be the new intersection point. The value of ϵ can be influenced by the user, but the default value is 0.3. If an element is intersected it is subdivided such that the intersection points and original nodes are connected such that new subelements arise in a natural way. These new subelements are considered to be either inside the obstacle or outside. In this way the "new" obstacle is formed by the connection of all intersection points. Since some intersection points have been moved to nodal points, this means that the new obstacle is some approximation of the original one.
Within the new obstacle we assume no flow (i.e. all elements are skipped). On the boundary of this obstacle, the velocity is set to zero. So in fact the problem is then solved as in Section 7.5.1.1.

The mesh for this example is completely identical to the mesh in Section 7.5.1.2.

The problem file has the shape

```
# obstacle04_1.prb
#
#  Problem file for 2d obstacle domain example
#  The problem considered here is that of a fixed obstacle in a fluid
#
#  In this example we compute the flow around the obstacle with a fixed mesh
#  The obstacle is defined as an obstacle and the internal velocities are
#  set to 0
#  The fluid mesh is adapted to the obstacle by computing the intersections
#  with the boundary of the obstacle (approximate adaptive method)
#  On the boundary of this intersection the velocities are set to 0
#
#  In this specific example we use Taylor-Hood linear triangles (mini element)
#  See Manual Examples Section 7.5.1
#
#  To run this file use:
#     sepcomp obstacle04_1.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
set warn off    ! suppress warnings
constants           # See Users Manual Section 1.4
   reals
      rho       = 1         # density of fluid
      eta       = 0.01      # dynamic viscosity
      u_obst    = 0         # u-velocity of obstacle
      v_obst    = 0         # v-velocity of obstacle
   vector_names
      velocity_pressure
end
debug_parameters
   plotobstacles
```

```
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
     elgrp1=903                # Type number for Navier-Stokes
                               # Taylor-Hood element

   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2

      degfd1, degfd2 = curves(c1)           # no-slip bottom wall
      degfd1, degfd2 = curves(c3)           # no-slip top wall
      degfd1, degfd2 = curves(c4)           # inlet
                       in_inner_obstacle 1  # For points that are only in
                                            # elements that are completely
                                            # in the obstacle both pressure and
                                            # velocity are given
      degfd1, degfd2 = in_boun_obstacle 1   # For the other points in the
                                            # obstacle only the velocity is
                                            # prescribed
      degfd1, degfd2 = on_boun_obstacle 1   # Points on the boundary get
                                            # prescribed velocity
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions
   curves(c4), degfd1, quadratic, max=1   # The u-component of the velocity at
                                          # inflow is parabolic
end

# input for non-linear solver
# See Users Manual Section 3.2.9

nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1d-4,print_level=2, lin_solver=1
   equation 1
      fill_coefficients 1
      change_coefficients
         at_iteration 2, sequence_number 1
         at_iteration 3, sequence_number 2
end

#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Sections 7.1

coefficients, problem=1
```

```
   elgrp1 ( nparm=20 )     # The coeffs are defined by 20 parameters
      icoef2 = 1           # type of constitutive equation (1=Newton)
      icoef5 = 0           # Type of linearization (0=Stokes flow)
      coef7  =  rho        # Density
      coef12 =  eta        #12:  Value of eta (dynamic viscosity)

end

#  Define the coefficients for the next iterations
#  See Users Manual Section 3.2.7

change coefficients, sequence_number = 1   # Input for iteration 2
   elgrp1
      icoef5 = 1             # 5:  Type of linearization (1=Picard iteration)
end

change coefficients, sequence_number = 2   # Input for iteration 3
   elgrp1
      icoef5 = 2             # 5:  Type of linearization (2=Newton iteration)
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is part is not necessary
#
structure                   # See Users Manual Section 3.2.3
 # Create the new temporary mesh by intersecting the obstacle with the fluid
 # mesh. The computations are carried out on this new mesh
   make_obstacle_mesh
 # Compute the velocity
   prescribe_boundary_conditions, velocity_pressure
   solve_nonlinear_system, velocity_pressure
   plot_vector velocity_pressure
   plot_contour velocity_pressure, degfd=3
 # Remove the temporary mesh and map the solution back
   remove_obstacle_mesh
 # Write the results to a file
   output
end
```

Figure 7.5.1.9 is a plot of the mesh with boundary nodes of the new obstacle marked with a coloured cross (orange inside, black boundary and red near the boundary). Elements inside the obstacle are coloured.
Figure 7.5.1.10 shows the velocity vectors.

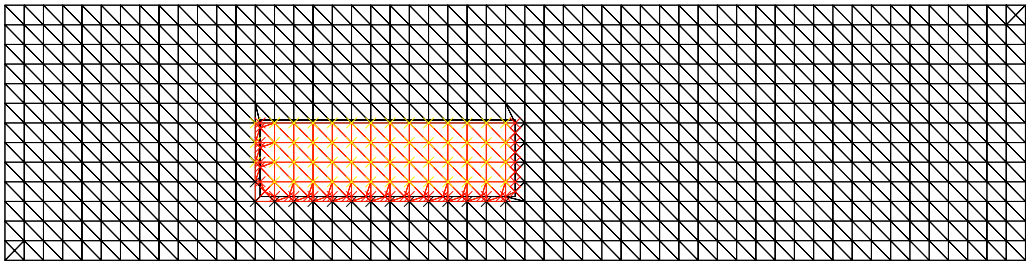


Figure 7.5.1.9: Adapted mesh

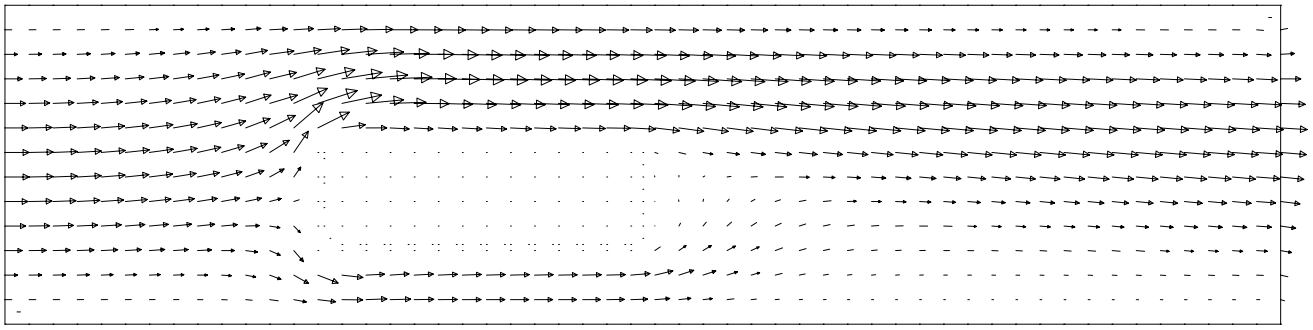


Figure 7.5.1.10: Velocity vectors

7.6 Stationary free surface flows

7.6.1 A simple extrusion problem: die-swell

The extrusion of a viscous incompressible jet from a die into an inviscid fluid is of considerable rheological importance. It is observed that far downstream the height of the extrudate is different from that of the die. This phenomenon is known as die-swell or extrudate swell. See for example Kruyt et al (1988). Here we assume that the jet is Newtonian and that the flow is steady and two-dimensional. Figure 7.6.1.1 The equations to be solved are the standard incompressible Navier-Stokes equations as described in Section 7.1 of the manual Standard Problems.
The boundary conditions on the fixed boundaries are as follows:

- symmetry axis: $u_2 = 0$, $\sigma^{12} = 0$ (symmetry condition)
- wall of die, except point P: $\mathbf{u} = 0$ (no slip condition)
- Point P (end of die), $u_2 = 0$, $\sigma^{12} = 0$, hence we have a slip condition in that point. Sometimes one also uses a no-slip condition in this point.
- inlet, $u_2 = 0$, u_1 prescribed by a quadratic velocity profile with maximum in symmetry axis and 0 on the wall

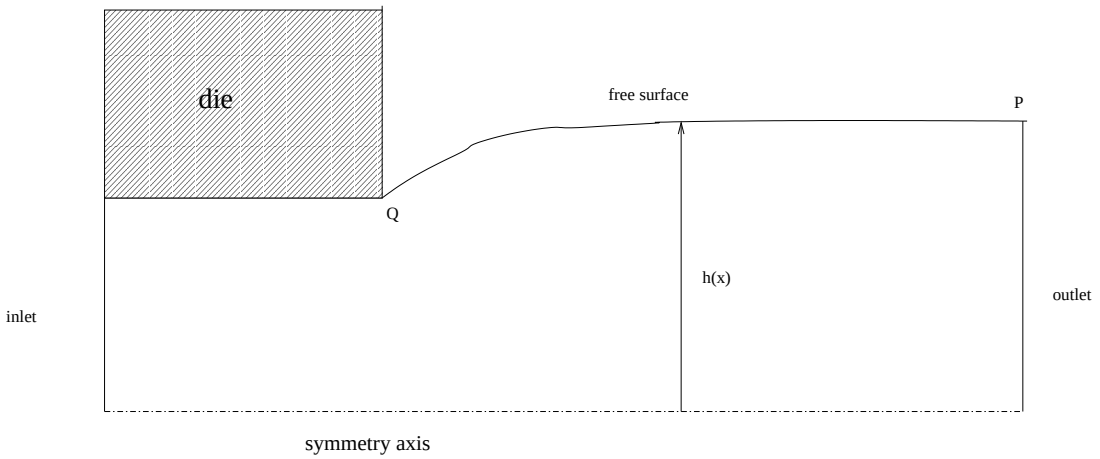


Figure 7.6.1.1: Geometry of the die-swell problem

- outlet, $u_2 = 0$, $\sigma^{11} = 0$, i.e parallel outflow and pressure at outflow is 0.

On the intersection point of die and free surface we have prescribed the normal velocity only. So in this specific point we allow slip. The reason is that mathematically speaking this is a singular point. Allowing some slip makes the singularity less pronounced, which means that grid refinement gives faster convergence.
From a physical point of view this is also a difficult point, since it is questionable if in the near surroundings of this point the continuum theory may be applied.

On the free boundary we need three boundary conditions:

$u_n = 0$, $\sigma^{nt} = 0$ and $\sigma^{nn} = \frac{\gamma}{R}$,

with γ the surface tension coefficient and $\frac{1}{R}$ the curvature of the free boundary. In other words the tangential stress is zero, the normal velocity is zero (no flow through the free surface) and the normal stress is prescribed by the surface tension and the zero pressure outside the fluid.

In this section we show the various methods treated in Section 7.6 of the manual Standard Problems to solve the die-swell problem.
To get these examples into your local directory use:

```
sepgetex dieswellxy
```

with x and y two independent one-digit numbers.
and to run it use:

```
sepmesh dieswellxy.msh
sepfree dieswellxy.prb
seppost dieswellxy.pst
```

After the each step you may view the results using sepview.

The following values for `x` are available:

```
x = 1, 2, 3
```

The following values for `y` are available:

```
y = 1, 2, 3, 4
```

The combinations 23 and 24 are not yet available.
Meaning of the various combinations:

x has the following meaning:

- **1** The free surface is adapted by one of the classical methods: the film method.
- **2** The free surface is adapted by the total linearization method.
- **3** The free surface is adapted by approximating the free boundary by a convection problem.

y has the following meaning:

- **1** A Cartesian coordinate system is assumed and in the end point of the die we have a slip condition.
- **2** A Cartesian coordinate system is assumed and in the end point of the die we have a no-slip condition.
- **3** An Axi-symmetric coordinate system is assumed and in the end point of the die we have a slip condition.
- **4** An Axi-symmetric coordinate system is assumed and in the end point of the die we have a no-slip condition.

Mark that we must apply sepfree instead of sepcomp, since the mesh is updated in each step.
In this example we have chosen to use one type of elements only (the extended quadratic Crouzeix-Raviart element) in combination with the penalty function method. The change to other types of elements or solution techniques for the incompressibility condition is very simple.

7.6.1.1 Die swell problem solved by the film method

The solution method for all free surface problems is the globally speaking the same. We start with an initial mesh, solve the Navier-Stokes equations and adapt the free surface as well as the mesh repeatedly in order to satisfy all boundary conditions on the free surface

For the initial mesh we assume that the free surface is a straight horizontal line starting in the point P.
The mesh used is defined by the following mesh input file.

```
#  dieswell11.msh
#
#  mesh file for die swell problem
#  See Manual Examples Section 7.6.1
#
#  To run this file use:
#     sepmesh dieswell.msh
#
#  Creates the file meshoutput
#
#  Define some general constants, they are stored in dieswell11.constants
#
include 'dieswell11.constants'
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1 = (  xleft,     ylow )       # Left-hand point on symmetry axis
      p2 = (  xwallend,  ylow )       # point on symmetry axis below end of die
      p3 = (  xright,    ylow )       # Right-hand point on symmetry axis
      p4 = (  xleft,     ytop )       # Left-hand point on die
      p5 = (  xwallend,  ytop )       # End point of die
      p6 = (  xright,    ytop )       # Right-hand point on free surface
      p7 = (  xbetween,  ylow )       # Point on the symmetry axis used to
                                      # define where the elements towards the
                                      # outflow may be enlarged
#
#  curves
#
   curves           # See Users Manual Section 2.3
    # Part of symmetry axis below the die:
      c1 = line  shape_cur ( p1, p2, nelm =  nelmh_die )
    # Rest of symmetry axis:
      c2 = curves(c8,c9)      # the line is splitted into 2 parts c8 and c9
    # Outflow boundary:
      c3 = translate c6 ( p3, p6 )   # translation of the inflow boundary
    # Die wall:
      c4 = translate c1 ( p4, p5 )
    # Free surface:
      c5 = translate c2 ( p5,-p6 )
    # Inflow boundary
      c6 = line  shape_cur ( p1, p4, nelm =  nelmv_die )
```

```
    # two parts of symmetry axis, below free surface
      c8 = line  shape_cur ( p2, p7, nelm =  nelmh_out )
      c9 = line  shape_cur ( p7, p3, nelm =  nelmh_far//
         ratio = 1, factor =  relax )
    # symmetry axis
      c10 = curves(c1,c2)
    # free surface
      c11 = curves(c4,c5)
    # Definition of all physical curves for use in computational program
      c inflow = curves(-c6)        # inflow boundary (from symmetry to top)
      c die = curves(c4)            # fixed wall (die)
      c free_surface = curves(c5)   # free surface
      c outflow = curves(c3)        # outflow boundary
      c symmetry = curves(c10)      # symmetry axis
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1 = rectangle  shape_sur (c10,c3,-c11,-c6)

   plot                           # make a plot of the mesh
                                  # See Users Manual Section 2.2
end
```

The mesh file uses an include file `dieswell11.constants` containing some constants that define parameters used in the mesh file and in the problem file. This include file has the following contents

```
#  dieswell11.constants
#
#  include file for the dieswell problem corresponding to dieswell11.msh
#  and dieswell11.prb
#
#  Contains constants that are used in the mesh generation and or computation
#
constants
   reals

     #  First parameters with respect to the mesh generation

      xright = 20              # end x-coordinate of free surface
      xleft = -3.5             # x-coordinate of inflow boundary
      ylow = 0                 # y-coordinate of symmetry axis
      ytop = 1                 # y-coordinate of die
      xwallend = 0             # end x-coordinate of die
      xbetween = 3.5           # Intermediate x-coordinate on
                               # free surface, used to define the subdivision
                               # of the free surface into elements
      relax = 3                # factor to define the subdivision
                               # of the free surface into elements

     #  Next some physical constants

      rho = 0.5                 # density of the fluid
      eta = 1                   # viscosity of the fluid
      gamma = 0.4               # surface tension
```

```
integers

  # Parameters to define the physical curves
   inflow = 20              # curve number of inflow boundary
   die = 21                 # curve number of die wall
   free_surface = 22        # curve number of free surface
   outflow = 23             # curve number of outflow boundary
   symmetry = 24            # curve number of symmetry axis
   die_point = 5            # User point number of end point of die
   fin_point = 6            # User point number of end point of free surface
  # Parameters to define the mesh
   shape_cur = 2            # quadratic elements along the curves
   shape_sur = 4            # quadratic triangles in the region
   irefine = 1              # Refinement parameter, is used as
                            # multiplication factor to define the number
                            # of elements along the various curves
   nelmh_die =  8* irefine  # Number of elements along the die
   nelmv_die =  4* irefine  # Number of elements along the inflow boundary
   nelmh_far =  4* irefine  # Number of elements on the first part of the
                            # free surface
   nelmh_out =  8* irefine  # Number of elements on the last part of the
                            # free surface
end
```

Figure 7.6.1.2 shows the curve numbers used in this example and Figure 7.6.1.3 the corresponding mesh.

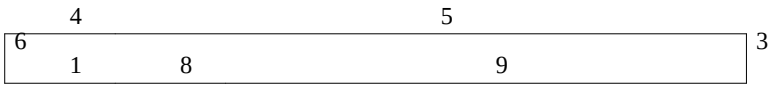


Figure 7.6.1.2: Curves for the die-swell problem

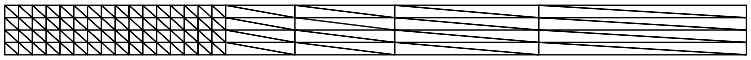


Figure 7.6.1.3: Mesh for the die-swell problem

To solve the free surface problem we start with a velocity vector $\mathbf{u} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, except on the boundaries where essential boundary conditions must be applied. In those boundaries we prescribe the correct boundary conditions. So the solution does not satisfy the condition $\mathbf{u} \cdot \mathbf{n} = \mathbf{0}$ on the free boundary.
After that the free surface algorithm is started. The boundary is updated so that the zero normal velocity boundary condition is approximated in a better way. This update is performed by applying

the film method with relaxation factor 1. This process is repeated until convergence is achieved. Finally the pressure is computed. Mark that in order to apply the film method it is necessary to start in the point Q of the free boundary, since there we have a zero displacement.
In order to apply the surface tension on the free boundary it is necessary to define boundary elements along the free surface. These boundary elements have type number 910, see the manual Standard Problems Section 7.1. In the starting point of the free boundary (point P) there is a zero displacement, so it is not necessary to prescribe the tangential direction in that point. However, in the end point Q, the normal displacement is not prescribed and therefore it is necessary to prescribe the tangential vector in that point. Since we assume that the outflow boundary is far enough we expect a horizontal free surface and we prescribe the tangential vector by $\mathbf{t} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. This is again done by a boundary element of type 910, which in this case reduces to a point element.
Mark that if the surface tension is zero, there is no need to give the boundary elements and also no need to give the tangential vector in the end point.
For simplicity we have only used the Picard linearization of the free boundary, but of course Newton linearization might be applied as well.
After running sepfree the mesh has been changed, which means that to test another update method it is necessary to rerun sepmesh.
The input file for program `sepfree` is given by

```
# dieswell11.prb
#
#  problem file for die swell problem
#  See Manual Examples Section 7.6.1
#
#  This is a stationary free surface problem
#  The velocity and pressure satisfy the non-linear Navier-Stokes equations
#  The Navier-Stokes equation is solved by a penalty function formulation
#  The free surface is updated in each step using the film method
#
#  To run this file use:
#     sepfree dieswell11.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#  Define some general constants, they are stored in dieswell11.constants
#
include 'dieswell11.constants'
#
#  Some specific constants are defined in this file
#
constants           # See Users Manual Section 1.4

   reals

   # Specific reals to be used for the computation

      penal = 1d-6               # penalty parameter

   # Names of vectors in the computation

   vector_names
      velocity            # velocity vector
```

```
      pressure            # pressure

end
#
#  Some information at the start of the computation
#
start
   norotate   # prevent rotation of plots
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=900               # Type number for Navier-Stokes, without swirl
                               # See Standard problems Section 7.1
   natboundcond            # Define the natural boundary conditions
                           # This is necessary to define the surface tension
      bngrp1=(type=910)    # Type number for natural boundary conditions
                           # for the Navier-Stokes
                           # See Standard problems Section 7.1
                           # In this case it concerns a line element for the
                           # surface tension
      bngrp2=(type=910)    # Type number for natural boundary conditions
                           # for the Navier-Stokes
                           # See Standard problems Section 7.1
                           # In this case it concerns a point element for the
                           # contact angle
   bounelements            # Define on which boundaries we have natural boundary
                           # conditions

        # the surface tension is defined on the free surface
        # the shape of the elements on the the curve is stored in shape_cur

      belm1 = curves(shape= shape_cur,c free_surface)
                                  # line elements along free surface
      belm2 = points(p fin_point)  # point element in end point
                                  # this element is needed in order to
                                  # prescribe the contact angle
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      curves 200 (c die)       # Fixed wall (die)
                               # all point on the die have no-slip condition
                               # except the last point (start of the free
                               # surface )
      degfd2, points(p die_point) # The normal velocity along the die is
                               # zero in the end point
      curves(c inflow)         # inflow, prescribed velocity
      degfd2,curves(c symmetry) # symmetry axis, only the y-velocity is 0
      degfd2,curves(c outflow)  # parallel outflow
end
```

```
#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because we have a free boundary problem

structure                   # See Users Manual Section 3.2.3

  # Create the start vector and fill the essential boundary conditions
   create_vector  velocity

  # Free surface iteration
   start_stationary_free_boundary

     # In each step the velocity is updated by solving a linear problem
     # The equation has been linearized by Picard
      solve_linear_system

   end_stationary_free_boundary

 # Compute the pressure
   derivatives, seq_deriv=1,  pressure

 # print the results
   print velocity, curves = (c free_surface)

 # Plot the velocity vector
   plot_vector velocity

 # Write results to sepcomp.out for postprocessing
   output

end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.10

create vector

   # Set the u velocity everywhere equal to 1 and the v velocity to 0
   degfd1, value = 1
   degfd2, value = 0

   # Set the u velocity on the die equal to 0
   degfd1, curves = c die, value = 0

   # The inflow velocity is a quadratic velocity profile for the x-component
   # Due to symmetry, the maximum is on the symmetry axis

   degfd1,curves(c inflow),half_quadratic, max=1.5

end

#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1
```

```
coefficients
   # First the Navier-Stokes elements
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
     icoef2 = 1            # Newtonian fluid
     icoef5 = 1            # Picard linearization
      coef6 =  penal       # penalty parameter
      coef7 =  rho         # density rho
      coef12=  eta         # viscosity eta
                           # Coefficients for the boundary elements
                           # First with respect to the line elements
                           # These are used for the surface tension
   bngrp1 (nparm=15)        # The coefficients for the natural boundary
                            # conditions are defined by 15 parameters
      icoef1 = 2            # iload (2=surface tension)
      coef6 =  gamma        # gamma (surface tension)
                           # Next with respect to the point element
                           # This is used to prescribe the contact angle
   bngrp2 (nparm=15)        # The coefficients for the natural boundary
                            # conditions are defined by 15 parameters
      icoef1 = 2            # iload (2=surface tension)
      coef6 =  gamma        # 6:  surface tension gamma
      coef7 = 1             # 7:  first component of tangential vector in end
                            #     point
      coef8= 0              # 8:  second component of tangential vector
end

# compute pressure
# See Users Manual, Section 3.2.11

derivatives
   icheld=7                          # icheld=7, pressure in nodes
                                     # See Standard problems Section 7.1
   seq_input_vector 1 =  velocity    # the pressure is computed form the velocity
end

# Information for the free surface computation
# See Users Manual Section 3.4.5

stationary_free_boundary
   maxiter = 20             # Maximum number of iterations
   miniter = 8              # Minimum number of iterations
                            # This is used to prevent that the process stops
                            # to early with a divergence message
   accuracy = 1d-3          # termination criterion
   print_level = 2          # Amount of output regarding the iteration process
   adapt_mesh = 1           # Defines the sequence number of the input block
                            # where it is described how the mesh must be adapted
   at_error = return        # If an error occurs a warning is issued, but the
                            # following statements are carried out
   write_mesh               # The final mesh is written to meshoutput
   criterion = relative     # Type of stopping criterion
end

# Information on how to adapt the mesh during the free surface iterations
```

```
# See Users Manual Section 3.4.3

adapt_mesh
   adapt_boundary = (1)     # Defines the sequence number of the input block
                            # where it is described how the boundary
                            # must be adapted
   plot_mesh                # Plot the mesh in each iteration step
end

# Information on how to adapt the boundary during the free surface iterations
# See Users Manual Section 3.4.4
# In this case we apply the film method without relaxation, i.e. factor = 1
# The fact that we use quadratic elements is utilized

adapt_boundary
   curves = (c free_surface)      # The free surface curve is adapted
   adaptation_method = film_method   # The method to be used is the film
                                  # method, see Users Manual 3.4.4
   quadratic                      # quadratic line elements are used
   plot_boundary                  # Plot the boundary in each iteration step
   factor=1                       # Multiplication factor (default)
end

end_of_sepran_input
```

The convergence of the free surface iteration process is very fast as can be seen in Table 7.6.1.1.

Table 7.6.1.1 Convergence of the free surface algorithm

Iteration	|| u(n) - u(n-1) ||
1	1.13E-02
2	1.12E-03
3	9.22E-05
4	3.38E-05
5	2.93E-06
6	9.61E-07
7	9.60E-08

Figures 7.6.1.4 and 7.6.1.5 show the final boundary and mesh. Intermediate pictures are almost the same. Postprocessing can be performed using for example the following post processing file

```
#  dieswell11.pst
#  Input file for postprocessing for die swell problem
#  See Manual Examples Section 7.6.1
#
#
#  To run this file use:
#     seppost dieswell11.pst > dieswell11.out
#
#  Reads the files meshoutput and sepcomp.out
#
#
postprocessing                    # See Users Manual Section 5.2

#
#  compute the stream function
```

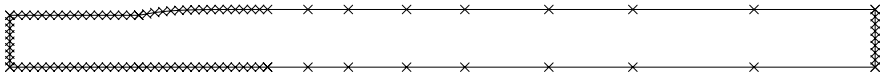


Figure 7.6.1.4: Boundary in the final iteration

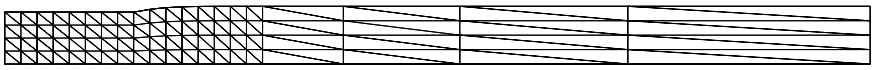


Figure 7.6.1.5: Mesh in the final iteration

```
#  See Users Manual Section 5.2
#  store in  stream_function

   compute  stream_function = stream function  velocity

#  Plot the results
#  See Users Manual Section 5.4

   plot vector  velocity              # Vector plot of velocity
   plot contour  pressure             # Contour plot of pressure
   plot coloured contour  pressure
   plot contour  stream_function      # Contour plot of stream function
   plot coloured contour  stream_function

end
```

Figure 7.6.1.6 shows the velocity vectors, Figure 7.6.1.7 the coloured pressure levels and Figure 7.6.1.8 the coloured stream function levels.

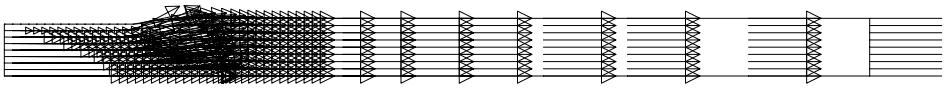


Figure 7.6.1.6: Velocity vectors in final mesh

Figure 7.6.1.7: Pressure levels in final mesh

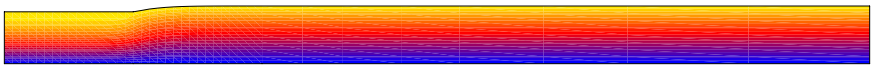


Figure 7.6.1.8: Stream function levels in final mesh

7.6.1.2 Die swell problem solved by the total linearization method

As an alternative to the classical Picard iterations of subsection 7.6.1.1 we demonstrate the total linearization method published in Kruyt et al (1988). This method does not only converge faster than the film method, it has also a larger convergence range. The mesh file, the constants file and the postprocessing file are completely identical to that of the film method. The only difference is in the computational part.
Again we start with a given velocity vector, however in order to apply the total linearization method it is also necessary to prescribe the initial pressure vector, since the pressure of the previous step is used in each iteration. Surface tension is part of the boundary element 915 and just as for the film method we have to prescribe the tangential vector in the end point of the free surface.
In this case the displacement of the free surface in y-direction is an unknown, which can be used in immediately to update the free boundary in each step.
The problem file used is given below

```
# dieswell21.prb
#
#  problem file for die swell problem
#  See Manual Examples Section 7.6.1
#
#  This is a stationary free surface problem
#  The velocity and pressure satisfy the non-linear Navier-Stokes equations
#  The Navier-Stokes equation is solved by a penalty function formulation
#  The update of the free surface is done by the method described in
#     N.P. Kruyt, C. Cuvelier, A. Segal, J. van der Zanden,
#     A total linearization method for solving viscous free boundary flow
#     problems by the finite element method,
#     Int. J. for Num. Methods in Fluids, Vol. 8, pp. 351-363, 1988
#  The Navier-Stokes equation is solved by the penalty function approach
#
#  To run this file use:
#     sepfree dieswell21.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#  Define some general constants, they are stored in dieswell21.constants
#
include 'dieswell21.constants'
#
#  Some specific constants are defined in this file
#
constants           # See Users Manual Section 1.4

   reals

   # Specific reals to be used for the computation

      penal = 1d-6              # penalty parameter

   # Names of vectors in the computation

   vector_names
      velocity            # velocity vector
      pressure            # pressure
```

```
end
#
#  Some information at the start of the computation
#
start
   norotate   # prevent rotation of plots
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=900                # Type number for Navier-Stokes, without swirl
                                # See Standard problems Section 7.1
   natboundcond                 # Define type numbers for elements corresponding
                                # to natural boundary conditions
                                # These are used to define the displacement of
                                # the free surface
      bngrp1=(type=915)         # Type number for special element used in the
      bngrp2=(type=915)         # total linearization method
   bounelements                 # Define the boundary elements for the total
                                # linearization method along the free surface
      belm1 = curves(c free_surface)  # line elements along free surface
      belm2 = points(p fin_point)     # point element in end point
                                 # this element is needed in order to
                                 # prescribe the contact angle
   essbouncond                   # Define where essential boundary conditions are
                                 # given (not the value)
                                 # See Users Manual Section 3.2.2
      curves 200 (c die)         # Fixed wall (die)
                                 # all point on the die have no-slip condition
                                 # except the last point (start of the free
                                 # surface )
      degfd2, points(p die_point) # The normal velocity along the die is
                                 # zero in the end point
      degfd3, points(p die_point) # The displacement of the free surface
                                 # in the end point of the die is 0
                                 # This is the third degree of freedom
                                 # along the free surface
      curves(c inflow)           # inflow, prescribed velocity
      degfd2,curves(c symmetry) # symmetry axis, only the y-velocity is 0
      degfd2,curves(c outflow)  # parallel outflow
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because we have a free boundary problem

structure                   # See Users Manual Section 3.2.3

  # Compute start vectors for the velocity and pressure
   create_vector,  velocity
```

```
   create_vector, sequence_number = 2,  pressure

  # Free surface iteration
   start_stationary_free_boundary_loop

      # first compute velocity, by solving system of linear equations
      solve_linear_system,  velocity

      # next compute pressure as derived quantity
      derivatives, seq_deriv = 1,  pressure

   end_stationary_free_boundary_loop

 # print the results
   print velocity, curves = (c free_surface)

 # Write results to sepcomp.out for postprocessing
   output

end

#
#  Define coefficients for the problem to be solved
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1
#
coefficients
   # First the Navier-Stokes elements
   elgrp 1 ( nparm = 20)  # The coefficients are defined by 20 parameters
      icoef2 = 1          # 2:  type of constitutive equation (1=Newton)
      icoef5 = 2          # 5:  Type of linearization (2=Newton)
      coef6 =  penal      # 6:  Penalty function parameter eps
      coef7 =  rho        # 7:  Density
      coef12=  eta        #12:  Value of eta (viscosity)

                          # Coefficients for the boundary elements
                          # First with respect to the line elements
                          # These are used for the surface tension
   bngrp 1 ( nparm = 10 )  # The coefficients are defined by 10 parameters
      coef6 =  rho         # 6:  density rho
      coef7 =  eta         # 7:  viscosity eta
      coef8 =  gamma       # 8:  surface tension gamma
      coef9 = old_vector  pressure   # pressure in previous iteration
                          # Next with respect to the point element
                          # This is used to prescribe the contact angle
   bngrp 2 ( nparm = 10 )  # The coefficients are defined by 10 parameters
      coef8 =  gamma       # 8:  surface tension gamma
      coef9 = 1            # 9:  first component of tangential vector in end
                           #     point
      coef10= 0            #10:  second component of tangential vector
end

#
#  Create start vector for velocity, including boundary conditions
#  See Users Manual Section 3.2.10
```

```
#
create vector
   degfd1, value = 1                # First set the u component equal to 1
                                    # Next set the inflow velocity
   degfd1, curves ( c inflow ), half_quadratic, max=1.5
   curves  ( c die ), value = 0  # On the die we have a no=slip condition
end
#
#  Create start vector for pressure
#
create vector, sequence_number = 2
   type = vector of special structure 6
   value = 0
end
#
# Definition of how to compute the derivatives (pressure)
#
derivatives
   icheld = 27                   # icheld=27, pressure in vertices
                                 # See Standard problems Section 7.1/7.6
   seq_input_vector =  velocity  # Defines the input vector (velocity)
end
#
# Information about free boundary problem, see Users Manual Section 3.4.5
#
stationary_free_boundary
   maxiter = 10             # Maximum number of iterations
   accuracy = 2d-4          # termination criterion
   print_level = 2          # Amount of output regarding the iteration process
   adapt_mesh = 1           # Defines the sequence number of the input block
                            # where it is described how the mesh must be adapted
   at_error = return        # If an error occurs a warning is issued, but the
                            # following statements are carried out
   write_mesh               # The final mesh is written to meshoutput
end
#
# Definition of how to adapt the mesh, see Users Manual Section 3.4.3
#
adapt_mesh
   adapt_boundary = (1)     # Defines the sequence number of the input block
                            # where it is described how the boundary
                            # must be adapted
   plot_mesh                # Plot the mesh in each iteration step
end
#
# Definition of how to adapt the boundary, see Users Manual Section 3.4.4
#
adapt_boundary
   curves = (c free_surface)      # The free surface curve is adapted
   adaptation_method = standard   # The method to be used is the standard
                                  # method: xnew = xold + alpha n
                                  # with alpha the third component of the
                                  # solution vector on the free surface
   quadratic                      # quadratic line elements are used
   number = 3                     # The third degree of freedom corresponds to
```

```
                                  # alpha
   plot_boundary                  # Plot the boundary in each iteration step
end

end_of_sepran_input
```

Convergence results can be found in Table 7.6.1.2.

Table 7.6.1.2 Convergence of the total linearization method

Iteration	|| u(n) - u(n-1) ||
1	7.67E-02
2	1.26E-02
3	3.75E-03
4	3.85E-04
5	3.01E-05

Of course the pictures are almost identical as those of the film method.

7.6.1.3 Die swell problem solved by approximating the free boundary by a convection problem

For this solution method we have to solve two problems per iteration. First we solve the Navier-Stokes equations and then a convection problem is solved to update the free boundary.
In each step we try to compute the stream line in Cartesian coordinates. in other words in each step the displacement with respect to the original boundary is computed. This means that we have to subtract the previous displacement in order to get the incremental displacement of the free boundary with respect to the present free boundary. In other words the algorithm reads

Start with a straight horizontal line as initial free boundary. This defines the initial region.
$k = 0$
Initialize the velocity $\mathbf{v}^0$
Initialize the total displacement $\mathbf{d}^0_{\text{tot}} = 0$
while not converged **do**
$k = k + 1$
Solve the flow problem on the present region using all but one of the boundary conditions on the free boundary. The result is the velocity $\mathbf{v}^k$
Solve the convection problem on the free boundary using $\mathbf{v}^k$ in order to compute $\mathbf{d}^k$.
Compute the displacement with respect to the present boundary: $\partial\mathbf{d} = \mathbf{d}^k$ - $\mathbf{d}^{k-1}_{\text{tot}}$.
Adapt the boundary using $\partial\mathbf{d}$.
Adapt the mesh by adapting the coordinates or if necessary by remeshing.
Compute the total displacement $\mathbf{d}^k_{\text{tot}} = \mathbf{d}^k = \mathbf{d}^{k-1}_{\text{tot}} + \partial\mathbf{d}$
end while

The intermediate steps of computing the total displacement and the displacement with respect to the present boundary are necessary since the program only updates the present boundary with a given displacement.
The solution of the convection problem along the free boundary is of course a scalar representing the new y-position of all points along the boundary. The x-position remains unchanged. This makes it necessary to map the scalar first into a vector with 2 components before computing the new free boundary. The problem in this case reads

```
#  dieswell31.prb
#
#  problem file for die swell problem
#  See Manual Examples Section 7.6.1
#
#  This is a stationary free surface problem
#  The velocity and pressure satisfy the non-linear Navier-Stokes equations
#  The Navier-Stokes equation is solved by a penalty function formulation
#  The free surface is updated in each step by solving a convection-diffusion
#  equation
#
#  To run this file use:
#     sepfree dieswell31.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#  Define some general constants, they are stored in dieswell31.constants
#
include 'dieswell31.constants'
#
#  Some specific constants are defined in this file
#
```

```
constants           # See Users Manual Section 1.4

   reals

   # Specific reals to be used for the computation

      penal = 1d-6              # penalty parameter
      kappa = 1d-10             # smoothing parameter for convection problem

   # Names of vectors in the computation

   vector_names
      velocity            # velocity vector
      pressure            # pressure
      y_displacement      # Displacement of the mesh in y-direction along the
                          # free surface with respect to the present mesh
      y_tot_displacement  # Displacement of the mesh in y-direction along the
                          # free surface with respect to the initial mesh
      displacement        # Displacement vector consisting of displacement
                          # vector in y-direction along free surface
                          # extended by zeros
                          # Is used in the update of the mesh
end
#
#  Some information at the start of the computation
#
start
   norotate   # prevent rotation of plots
end
#
#  Define the type of problem to be solved
#
problem 1                 # See Users Manual Section 3.2.2
                          # Problem 1 refers to the Navier-Stokes equations

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=900               # Type number for Navier-Stokes, without swirl
                               # See Standard problems Section 7.1
   natboundcond           # Define the natural boundary conditions
                          # This is necessary to define the surface tension
      bngrp1=(type=910)   # Type number for natural boundary conditions
                          # for the Navier-Stokes
                          # See Standard problems Section 7.1
                          # In this case it concerns a line element for the
                          # surface tension
      bngrp2=(type=910)   # Type number for natural boundary conditions
                          # for the Navier-Stokes
                          # See Standard problems Section 7.1
                          # In this case it concerns a point element for the
                          # contact angle
   bounelements           # Define on which boundaries we have natural boundary
                          # conditions

        # the surface tension is defined on the free surface
```

```
        # the shape of the elements on the the curve is stored in shape_cur

      belm1 = curves(shape= shape_cur,c free_surface)
                                  # line elements along free surface
      belm2 = points(p fin_point)  # point element in end point
                                  # this element is needed in order to
                                  # prescribe the contact angle

   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      curves 200 (c die)        # Fixed wall (die)
                                # all point on the die have no-slip condition
                                # except the last point (start of the free
                                # surface )
      degfd2, points(p die_point) # The normal velocity along the die is
                                # zero in the end point
      curves(c inflow)          # inflow, prescribed velocity
      degfd2,curves(c symmetry) # symmetry axis, only the y-velocity is 0
      degfd2,curves(c outflow)  # parallel outflow

problem 2                 # refers to the solution of the convection problem
                          # along the free surface

   types                  # Define types of elements,
      elgrp1=(type=0)     # Since the convection problem is only solved
                          # along the free boundary we use type number 0
                          # in the inner region
   natboundcond           # Define the natural boundary conditions
                          # In this case this actually the equation
      bngrp1=(type=800)   # Type number for the convection equation
                          # See Standard problems Section 3.1
   bounelements           # Define on which boundaries we have to solve
                          # convection problem
      belm1=curves(shape=1,c free_surface)  # Only on the free surface
                          # in this case we use linear elements
   essbouncond            # Define where essential boundary conditions are
                          # given
      points = p die_point  # The displacement at the first point of the
                          # free surface is 0
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because we have a free boundary problem

structure                   # See Users Manual Section 3.2.3

  # Create the start vector for the velocity and
  # fill the essential boundary conditions
   create_vector  velocity

  # Initialize the total displacement vector (0)
   create_vector, sequence_number=2,  y_tot_displacement
```

```
  # Free surface iteration
   start_stationary_free_boundary_loop

     # first compute velocity, by solving system of linear equations
      solve_linear_system, velocity

     # Next we compute the displacement of the free surface by solving a
     # convection equation
     # First the essential boundary condition is stored
      prescribe_boundary_conditions,  y_displacement

     # Next the system of equations is solved, this results in the
     # displacement with respect to the initial mesh
      solve_linear_system, seq_coef = 2,  y_displacement, problem = 2

     # The displacement in the previous iterations must be subtracted
     # from this displacement in order to get the displacement
     # with respect to the present mesh
       y_displacement =  y_displacement- y_tot_displacement

     # Map y_displacement into displacement vector in order to use
     # this vector in the update of the free surface
     # This is necessary since the update algorithm expects a velocity vector
     # consisting of 2 components instead of 1

       displacement = map y_displacement, type = 0, degfd = 2

     # Finally the total displacement is updated
       y_tot_displacement =  y_displacement +  y_tot_displacement

   end_stationary_free_boundary_loop

 # Compute the pressure
   derivatives, seq_coef=1, seq_deriv=1,  pressure

 # print the results
   print velocity, curves = (c free_surface)

 # Write results to sepcomp.out for postprocessing
   output

end

#  The essential boundary condition for the convection equation is zero
#  See Users Manual Section 3.2.5

essential boundary conditions, problem 2
   value = 0
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.10

create vector, problem 1
```

```
   degfd1, value = 1                # First set the u component equal to 1
                                    # Next set the inflow velocity
   degfd1, curves ( c inflow ), half_quadratic, max=1.5
   curves  ( c die ), value = 0  # On the die we have a no=slip condition
end

#  Initialize the total y-displacement vector

create vector, problem 2, sequence_number = 2
   value = 0
end

#  coefficients for velocity problem (Navier-Stokes)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   # First the Navier-Stokes elements
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
     icoef2 = 1            # Newtonian fluid
     icoef5 = 1            # Picard linearization
      coef6 =  penal       # penalty parameter
      coef7 =  rho         # density rho
      coef12=  eta         # viscosity eta
                          # Coefficients for the boundary elements
                          # First with respect to the line elements
                          # These are used for the surface tension
   bngrp1 (nparm=15)       # The coefficients for the natural boundary
                           # conditions are defined by 15 parameters
      icoef1 = 2           # iload (2=surface tension)
      coef6 =  gamma       # gamma (surface tension)
                          # Next with respect to the point element
                          # This is used to prescribe the contact angle
   bngrp2 (nparm=15)       # The coefficients for the natural boundary
                           # conditions are defined by 15 parameters
      icoef1 = 2           # iload (2=surface tension)
      coef6 =  gamma       # 6:  surface tension gamma
      coef7 = 1            # 7:  first component of tangential vector in end
                           #     point
      coef8= 0             # 8:  second component of tangential vector
end

#  coefficients for convection diffusion problem (displacement of boundary)
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1/7.6

coefficients, sequence_number = 2
   bngrp1 (nparm=20)
      icoef2 = 1                # first order upwind
      icoef5 = 4                # transformation
      coef6 =  kappa            # diffusion parameter, used for smoothing
      coef12= old_vector  velocity, degfd1   # velocity
      coef16= old_vector  velocity, degfd2   # right-hand side
end

# compute pressure
```

```
# See Users Manual, Section 3.2.11

derivatives
   icheld=7                          # icheld=7, pressure in nodes
                                     # See Standard problems Section 7.1
   seq_input_vector 1 =  velocity    # the pressure is computed form the velocity
end
#
# Information about free boundary problem, see Users Manual Section 3.4.5
#
stationary_free_boundary
   maxiter = 10               # Maximum number of iterations
   accuracy = 2d-4           # termination criterion
   print_level = 2           # Amount of output regarding the iteration process
   adapt_mesh = 1            # Defines the sequence number of the input block
                             # where it is described how the mesh must be adapted
   at_error = return         # If an error occurs a warning is issued, but the
                             # following statements are carried out
   seq_vectors = displacement  # Defines the "velocity" vector to be used
                             # when updating the boundary
   write_mesh                # The final mesh is written to meshoutput
end
#
# Definition of how to adapt the mesh, see Users Manual Section 3.4.3
#
adapt_mesh
   adapt_boundary = 1        # Defines the sequence number of the input block
                             # where it is described how the boundary
                             # must be adapted
   plot_mesh                 # Plot the mesh in each iteration step
end
#
# Definition of how to adapt the boundary, see Users Manual Section 3.4.4
#
adapt_boundary
   curves = (c free_surface)      # The free surface curve is adapted
   adaptation_method = velocity   # The method to be used is the standard
                                  # method: xnew = xold + v
                                  # where v is the velocity vector
   quadratic                      # quadratic line elements are used
   plot_boundary                  # Plot the boundary in each iteration step
end

end_of_sepran_input
```

Convergence results can be found in Table 7.6.1.3.

Table 7.6.1.3 Convergence of the convection approach

Iteration	|| u(n) - u(n-1) ||
1	1.47E-01
2	2.68E-02
3	3.67E-03
4	2.26E-04
5	1.44E-04

Again the pictures are almost identical as those of the film method.

7.6.2 Shape of a drop under the influence of surface tension

In this section we demonstrate the effect of surface tension. The example has been provided by Frank Dammel of the Technical University of Darmstadt (Germany). A classical test example is that of a drop in a fluid in rest. In this stationary case we start with a drop of arbitrary shape, in this particular example a square. The only force in the flow is the surface tension acting on the drop. Due to this surface tension the drop must take the shape of a circle and the flow must be at rest. The pressure inside the drop must be constant with value equal to the surface tension coefficient γ.
Due to symmetry it is sufficient to take only one quarter of the drop.

To get these examples into your local directory use:

```
sepgetex fs_drop_testxx
```

where xx may be either 11 (Cartesian case) or 12 (Axi-symmetric case), and to run it use:

```
sepmesh fs_drop_testxx.msh
sepfree fs_drop_testxx.prb
seppost fs_drop_testxx.pst
```

The initial mesh consists of a square. In order to prevent the necessity of remeshing during the iterations the mesh consists of a fixed inner square and an outer part of which the shape is adapted during each iteration. Figure 7.6.2.1 shows the subdivision of the regions and the definition of the curves. The initial mesh may be created by the following input file

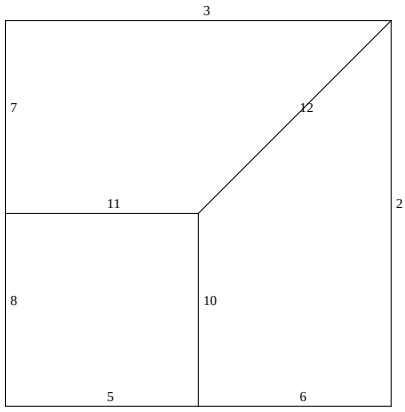


Figure 7.6.2.1: Definition of the curves in the surface tension test example

```
#  fs_drop_test11.msh
#
#  mesh file for testing the surface tension on a drop in a no-flow region
#  See Manual Examples Section 7.6.2
#
#  To run this file use:
#     sepmesh fs_drop_test11.msh
#
#  Creates the file meshoutput
#
#  Define some general constants, they are stored in fs_drop_test11.constants
#
```

```
include 'fs_drop_test11.constants'
#
#  Define the mesh
#
mesh2d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1 = (0,0)                    # Centre of drop
      p2 = ( length,0)              # end point of starting rectangle in x-dir
      p3 = ( length, width)         # right upper point of starting rectangle
      p4 = (0, width)               # end point of starting rectangle in y-dir
      p5 = (  length/2 ,0)          # middle point on lower boundary
      p6 = (0,  width/2 )           # middle point on left-hand side boundary
      p7 = (  length/2 ,  width/2 ) # centre point of region
#
#  curves
#
   curves           # See Users Manual Section 2.3

      c1 = curves(c5,c6)                        # lower boundary of rectangle
                                                # subdivided into 2 parts
      c5 = line  shape_cur ( p1,p5, nelm= n )   # left-hand side part
                                                # of lower boundary
      c6 = line  shape_cur ( p5,p2, nelm= n )   # right-hand side part
                                                # of lower boundary
      c2 = line  shape_cur ( p2,p3, nelm= m )   # right-hand side of rectangle
      c3 = line  shape_cur ( p3,p4, nelm= n )   # upper boundary of rectangle
      c4 = curves(c7,c8)                        # right-hand side of rectangle
      c7 = line  shape_cur ( p4,p6, nelm= m )   # upper part of
                                                # left-hand side of rectangle
      c8 = line  shape_cur ( p6,p1, nelm= m )   # lower part of
                                                # left-hand side of rectangle
      c10= line  shape_cur ( p5,p7, nelm= n )   # right-hand side of inner
                                                # rectangle
      c11= line  shape_cur ( p7,p6, nelm= m )   # upper side of inner
                                                # rectangle
      c12= line  shape_cur ( p7,p3, nelm= m )   # line from centre to point
                                                # at top and right-hand side
      c free_surface = curves(c2,c3)
      c symm_hor = curves(c1)
      c symm_vert = curves(c4)
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
      s1 = rectangle  shape_sur (c5,c10,c11,c8)   # inner rectangle
      s2 = rectangle  shape_sur (c6,c2,-c12,-c10) # right-hand side quad
      s3 = rectangle  shape_sur (c3,c7,-c11,c12)  # upper quad

   plot                          # make a plot of the mesh
                                 # See Users Manual Section 2.2

end
```

The mesh file uses an include file `fs_drop_test11.constants` containing some constants that define parameters used in the mesh file and in the problem file. This include file has the following contents

```
#  fs_drop_test11.constants
#
#  include file for the fs_drop_test problem corresponding to fs_drop_test11.msh
#  and fs_drop_test11.prb
#
#  Contains constants that are used in the mesh generation and or computation
#
constants
   reals

     #  First parameters with respect to the mesh generation

      length = 1               # length of starting rectangle
      width = 1                # width of starting rectangle

     #  Next some physical constants

      rho = 1                   # density of the fluid
      eta = 1                   # viscosity of the fluid
      gamma = 0.1               # surface tension

   integers

     # Parameters to define the physical curves
      free_surface = 20        # curve number of free surface
      symm_hor = 21            # horizontal symmetry axis
      symm_vert = 22           # vertical symmetry axis
     # Parameters to define the mesh
      shape_cur = 2            # quadratic elements along the curves
      shape_sur = 4            # quadratic triangles in the region
      n = 2                    # Number of elements in x-direction
      m = 2                    # Number of elements in y-direction
      jcart = 0                # Defines type of coordinate system
                               # 0 = Cartesian
                               # 1 = Axi-symmetric
end
```

Figure 7.6.2.2 shows the initial mesh. Since in this example there is no flow we have chosen to solve the Stokes equations only. On the symmetry axis we have the standard symmetry boundary conditions and on the free surface we impose the given surface tension as well as a zero shear stress. In each iteration the boundary is updated using the computed normal velocity, so that in the end the boundary condition $\mathbf{u} \cdot \mathbf{n} = 0$ is satisfied. This method converges linearly and not very fast. Approximately 25 iterations were necessary to reach the final shape.

```
# fs_drop_test11.prb
#
#  problem file for testing the surface tension on a drop in a no-flow region
#  See Manual Examples Section 7.6.2
#
#  This is a stationary free surface problem
#  The velocity and pressure satisfy the linear Stokes equations
#  The Stokes equation is solved by a penalty function formulation
```

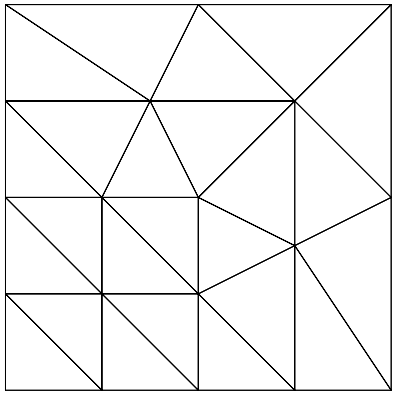


Figure 7.6.2.2: Initial mesh for the surface tension test example

```
#  The free surface is updated in each step using the normal_velocity
#  computed in the previous iteration
#
#  To run this file use:
#     sepfree fs_drop_test11.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#  Define some general constants, they are stored in fs_drop_test11.constants
#
include 'fs_drop_test11.constants'
#
#
#
#  Some specific constants are defined in this file
#
constants           # See Users Manual Section 1.4

   reals

   # Specific reals to be used for the computation

      penal = 1d-6              # penalty parameter

   # Names of vectors in the computation

   vector_names
      velocity            # velocity vector
      pressure            # pressure


end
#
#  Some information at the start of the computation
#
start
```

```
   norotate   # prevent rotation of plots
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=900               # Type number for Navier-Stokes, without swirl
                               # See Standard problems Section 7.1
   natboundcond             # Define the natural boundary conditions
                            # This is necessary to define the surface tension
      bngrp1=(type=910)     # Type number for natural boundary conditions
                            # for the Navier-Stokes
                            # See Standard problems Section 7.1
                            # In this case it concerns a line element for the
                            # surface tension
   bounelements             # Define on which boundaries we have natural boundary
                            # conditions

        # the surface tension is defined on the free surface
        # the shape of the elements on the the curve is stored in shape_cur

      belm1 = curves(shape= shape_cur,c free_surface)
                                  # line elements along free surface
   essbouncond                 # Define where essential boundary conditions are
                               # given (not the value)
                               # See Users Manual Section 3.2.2
      degfd2, curves(c symm_hor)  # symmetry in horizontal direction (u_y=0)
      degfd1, curves(c symm_vert) # symmetry in vertical direction (u_x=0)
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This is necessary because we have a free boundary problem

structure                  # See Users Manual Section 3.2.3

  # Create the start vector and fill the essential boundary conditions
   create_vector  velocity

  # Free surface iteration
   start_stationary_free_boundary

     # In each step the velocity is updated by solving a linear problem
      solve_linear_system, velocity

   end_stationary_free_boundary

 # Compute the pressure
   derivatives, pressure

 # print the results
   print velocity, curves = (c free_surface)
```

```
 # Plot the velocity vector
   plot_vector, velocity

 # Write results to sepcomp.out for postprocessing
   output

end

#  Define the coefficients for the problems
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 7.1

coefficients
   # First the Navier-Stokes elements
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
     icoef2 = 1            # Newtonian fluid
     icoef4 =  jcart       # Type of coordinate system
     icoef5 = 1            # Picard linearization
      coef6 =  penal       # penalty parameter
      coef7 =  rho         # density rho
      coef12=  eta         # viscosity eta
                           # Coefficients for the boundary elements
                           # First with respect to the line elements
                           # These are used for the surface tension
   bngrp1 (nparm=15)        # The coefficients for the natural boundary
                            # conditions are defined by 15 parameters
      icoef1 = 2            # iload (2=surface tension)
      icoef4 =  jcart       # Type of coordinate system
      coef6 =  gamma        # gamma (surface tension)
                           # Next with respect to the point element
                           # This is used to prescribe the contact angle
end

# compute pressure
# See Users Manual, Section 3.2.11

derivatives
   icheld=7                         # icheld=7, pressure in nodes
                                    # See Standard problems Section 7.1
   seq_input_vector 1 =  velocity   # the pressure is computed form the velocity
end

# Information for the free surface computation
# See Users Manual Section 3.4.5

stationary_free_boundary, sequence_number = 1
   maxiter = 50             # Maximum number of iterations
   miniter = 1              # Minimum number of iterations
   accuracy = 1d-4          # termination criterion
   print_level = 2          # Amount of output regarding the iteration process
   adapt_mesh = 1           # Defines the sequence number of the input block
                            # where it is described how the mesh must be adapted
   at_error = return        # If an error occurs a warning is issued, but the
                            # following statements are carried out
```

```
   write_mesh                # The final mesh is written to meshoutput
   criterion = absolute      # Type of stopping criterion
                             # Since the velocity itself goes to zero,
                             # it is necessary to use an absolute criterion
end

# Information on how to adapt the mesh during the free surface iterations
# See Users Manual Section 3.4.3

adapt_mesh
   adapt_boundary = (1)      # Defines the sequence number of the input block
                             # where it is described how the boundary
                             # must be adapted
   plot_mesh                # Plot the mesh in each iteration step
end


# Information on how to adapt the boundary during the free surface iterations
# See Users Manual Section 3.4.4
# In this case we apply the film method without relaxation, i.e. factor = 1
# The fact that we use quadratic elements is utilized

adapt_boundary
   curves = (c free_surface)      # The free surface curve is adapted
   adaptation_method = normal_velocity,
                                  # the computed normal velocity is
                                  # used, to estimate the new surface
                                  # see Users Manual 3.4.4
   quadratic                      # quadratic line elements are used
   plot_boundary                  # Plot the boundary in each iteration step
   factor=1                       # Multiplication factor (default)
                                  # The next two lines force the begin and
                                  # end point to remain on the symmetry axis
   exclude_begin=second           # x2-coord. of first node remains unchanged
   exclude_end=first              # x1-coord. of last node remains unchanged
end

end_of_sepran_input
```

Figure 7.6.2.3 shows the boundary of the region during all the iterations and Figure 7.6.2.4 the final mesh. The postprocessing file for this example is not special and is not repeated in this manual.

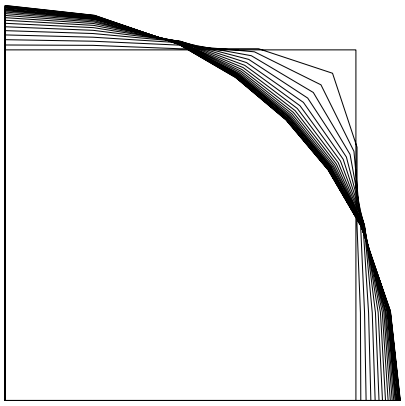


Figure 7.6.2.3: Boundary during the iteration process

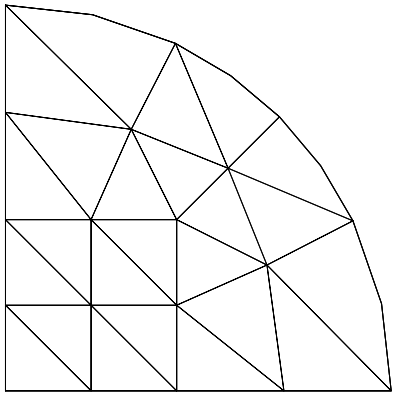


Figure 7.6.2.4: Final mesh for the surface tension test example

8 Second order elliptic and parabolic equations using spectral elements

8.1 Second order real linear elliptic and parabolic equations with one degree of freedom

8.1.1 Example of a 1D convection-diffusion problem by spectral elements

In this section an artificial example of the solution of a convection-diffusion equation with inhomogeneous Dirichlet boundary conditions is considered. The purpose of this example is to show how the spectral elements in this chapter may be used, and to compare some results of the spectral element method with the finite element method. The filling of the coefficients is also handled in this example.
Consider the following problem :

$$\begin{array}{ll} -\Delta c + (u \cdot \nabla)c = f & \text{on } [-1,1] \\ c = g & \text{in } x = -1, \text{ and } x = 1, \end{array} \tag{8.1.1.1}$$

where $u = \tan(x)$ and $f = 2\sin(x)$. The exact solution is then given by $c = \sin(x)$.

To discretize the domain of this problem, we use two spectral elements of tenth order, see Figure 8.1.1.1

MESH

Figure 8.1.1.1: Spectral element mesh consisting of two elements of tenth order

To create this mesh the following input is used :

```
************************************************************************
*
*      File:  exam8-1-1a.msh
*
*      Contents:  Mesh for the example 8-1-1 in the manual standard problems
*                 Line (0,1)
*                 2 spectral elements of tenth order
*
************************************************************************
*
mesh1d
  points
    p1=0.0d0
    p2= 1.0d1
  curves
    c1=line1(p1,p2,nelm=2)
  intermediate points
    sidepoints=9,subdivision=legendre,midpoints=filled
  plot(jmark=3, numsub=1)
end
```

The internal spectral elements are defined by the type number 600. Only the coefficients 6, 12 and 16 are defined. Coefficient 6 gets the value 1, where coefficients 12 and 16 define the functions u and f respectively. They are defined using the function subroutine FUNCCF
The boundary conditions at the points p1 and p2 are essential boundary conditions. The are defined using the function subroutine FUNCBC. To compare the numerical solution with the exact solution an additional function FUNC is defined which contains the exact solution. The following main program is used

```
c ************************************************************************
c
c      File:  exam8-1-1a.f
c
c      Contents:  Main program for the test example described
c                 in the SEPRAN manual standard problems 8-1-1
```

```
c                Artificial analytical example
c                This program uses the most simple version
c                Since a function subroutine is used for the solution,
c                it is not possible to used sepcomp
c
c      Usage:    Compile and link this program with the SEPRAN libraries
c                seplink exam8-1-1
c
c                Run this program with input: exam8-1-1a.prb or
c                                             exam8-1-1b.prb
c
c                exam8-1-1 < exam8-1-1a.prb > exam8-1-1a.out  or
c                exam8-1-1 < exam8-1-1b.prb > exam8-1-1b.out
c
c      version 1.0    date   24-07-96
c
c ***********************************************************************
c
      program exam811
      implicit none
      call sepcom(0)
      end

c     --- function funcbc for the definition of boundary conditions


      double precision function funcbc(ichois,x,y,z)
      implicit none
      integer ichois
      double precision x,y,z
      if (ichois.eq.1) funcbc = dsin(x)
      end

c     --- function func for the definition of exact solution

      double precision function func(ichois,x,y,z)
      implicit none
      integer ichois,k,l
      double precision x,y,z
c
      if (ichois.eq.1) func = dsin(x)
      end


c     --- function funccf for the definition of the coefficients


      double precision function funccf(ichois,x,y,z)
      implicit none
      integer ichois
      double precision x,y,z

      if (ichois.eq.1) funccf= dtan(x)
      if (ichois.eq.2) funccf= 2*sin(x)
      end
```

The resulting part of the input file then reads

```
************************************************************************
*
*      File:  exam8-1-1a.prb
*
*      Contents:  Input for program exam8-1-1 described in Section 8.1.1 in
*                 the manual standard problems
*                 Artificial analytical example
*                 The standard sepcomp approach is used
*
************************************************************************
*
*  Problem definition

problem
  types
  elgrp1 = ( type = 600)
 essbouncond
  degfd1=points(p1,p2)
end

essential boundary conditions
  degfd1=points(p1,p2) func=1
end

coefficients
 elgrp1(nparm=20)
 coef6 = 1.0d0
 coef12= (func=1)
 coef16= (func=2)
end
```

Once the solution has been computed, it may be printed and plotted by the post processing program SEPPOST. The input file requires be SEPPOST is given below :

```
************************************************************************
*
*      File:  exam8-1-1.pst
*
*      Contents:  Input for the post processing part of the example described
*                 in Section 8.1.1 of the manual standard problems
*                 Artificial analytical example
*
*      Usage:     seppost exam8-1-1.pst > exam8-11.out
*
************************************************************************
*
post processing
   name v0 = solution
   print v0
   plot function v0
end
```

Number of elements	Order	Degrees of freedom	Error
1	10	11	6.12E-03
1	15	16	1.96E-06
1	20	21	2.75E-11
2	10	21	5.43E-06
2	15	31	7.82E-11
2	20	41	2.27E-14
10	1	11	2.12E-01
15	1	16	9.65E-02
20	1	21	5.22E-02
30	1	31	2.25E-02
40	1	41	1.12E-02
100	1	101	2.02E-03
500	1	501	8.96E-05
1000	1	1001	1.66E-05

Table 8.1.1.1: Comparison between high order spectral and low order finite elements

The input file prints and plots the computed solution. Figure 8.1.1.2 shows the plot made by the program SEPPOST. This plot is visualized by the program SEPVIEW.

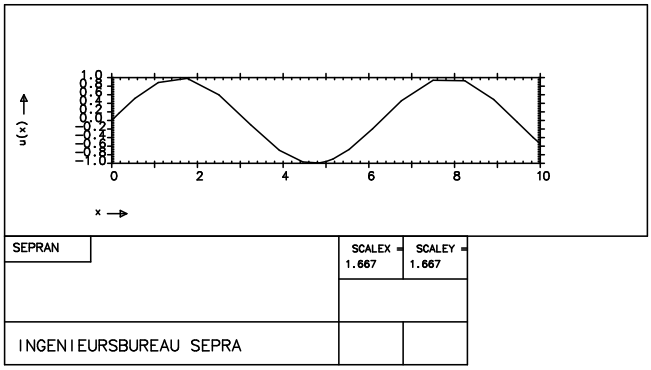


Figure 8.1.1.2: Spectral element solution of example 8-1-1

It is also possible to use element 800 to solve this problem. If the user wishes to do so, only the mesh and in the input file should be changed (type = 600, should be set to type = 800). The coefficients are the same. To demonstrate the spectral accuracy some examples are presented in Table 8.1.1.1.

8.1.4 Example of a 3D Helmholtz problem by spectral elements

In this section an artificial example of the solution of a Helmholtz equation with inhomogeneous Dirichlet boundary conditions is considered. The purpose of this example is to show how the spectral elements in this chapter may be used. The linear system of equations after discretization is solved using a finite element preconditioned conjugate gradient method.
This example is available in three versions. To get these versions into your local directory use:

```
sepgetex exam8-1-4x
```

with x: nothing a or b

and to run it use:

```
sepmesh exam8-1-4x.msh
seplink exam8-1-4x
exam8-1-4x < exam8-1-4x.prb
```

Consider the following problem:

$$\begin{array}{ll} -\Delta c + c = f & \text{on } \Omega = [-1,1]\text{x}[-1,1]\text{x}[0,4] \\ c = g & \text{in } \partial\Omega \end{array} \tag{8.1.4.1}$$

where $f = 4\sin(x)\sin(y)\sin(z)$. The exact solution is then given by $c = \sin(x)\sin(y)\sin(z)$.

To discretize the domain of this problem, we use eight spectral elements of eighth order, see Figure 8.1.4.1

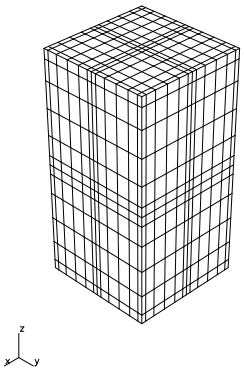


Figure 8.1.4.1: Finite element representation of a spectral element mesh consisting of eight elements of eighth order

Version `exam8-1-4` uses in principle program sepcomp, however since function subroutines must be supplied the most simple main program is provided. The FEM preconditioner uses a direct solver, i.e. the matrix is "inverted" by an LU decomposition.
Version `exam8-1-4a` is almost identical to `exam8-1-4`. The only difference is that also the finite element preconditioner is itself an iterative method. So the preconditioned system is solved approximately.
In version `exam8-1-4b` a user written main program is used.

The mesh is created by program `sepmesh` with the following input:

```
#  exam8-1-4.msh
#
```

```
#  mesh file for example 8.1.4, Helmholtz equation with a spectral element
#  method (3D)
#
#  To run this file use:
#     sepmesh exam8-1-4.msh
#
#  Creates the file meshoutput
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      x_low = -1       # lower value of x
      x_upp =  1       # upper value of x
      y_low = -1       # lower value of y
      y_upp =  1       # upper value of y
      z_low =  0       # lower value of z
      z_upp =  4       # upper value of z
   integers
      n = 2            # number of elements in length direction
      m = 2            # number of elements in width direction
      l = 2            # number of elements in width direction
      lin = 1          # linear line elements
      sur = 5          # bi-linear quadrilaterals
      vol =13          # tri-linear hexahedrons
      nside = 5        # order of spectral elements
end
#
#  Define the mesh
#
mesh3d              # See Users Manual Section 2.2
#
#  user points
#
   points           # See Users Manual Section 2.2
      p1=($x_low,$y_low,$z_low)   # Left under point bottom surface
      p2=($x_upp,$y_low,$z_low)   # Right under point bottom surface
      p3=($x_upp,$y_upp,$z_low)   # Right upper point bottom surface
      p4=($x_low,$y_upp,$z_low)   # Left upper point bottom surface
      p5=($x_low,$y_low,$z_upp)   # Left under point top surface
      p6=($x_upp,$y_low,$z_upp)   # Right under point top surface
      p7=($x_upp,$y_upp,$z_upp)   # Right upper point top surface
      p8=($x_low,$y_upp,$z_upp)   # Left upper point top surface
#
#  curves
#
   curves           # See Users Manual Section 2.3

      c1  = line $lin ( p1,p2,nelm=$n )   # bottom surface
      c2  = line $lin ( p2,p3,nelm=$m )
      c3  = line $lin ( p3,p4,nelm=$n )
      c4  = line $lin ( p4,p1,nelm=$m )
      c5  = line $lin ( p5,p6,nelm=$n )   # top surface
      c6  = line $lin ( p6,p7,nelm=$m )
```

```
      c7  = line $lin ( p7,p8,nelm=$n )
      c8  = line $lin ( p8,p5,nelm=$m )
      c9  = line $lin ( p2,p6,nelm=$l )   # lines from top to bottom
      c10 = line $lin ( p3,p7,nelm=$l )
      c11 = line $lin ( p4,p8,nelm=$l )
      c12 = line $lin ( p1,p5,nelm=$l )
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4

      s1 = rectangle $sur (c1,c2,c3,c4)         # bottom surface
      s2 = rectangle $sur (c5,c6,c7,c8)         # top surface
      s3 = rectangle $sur (c1,c9,-c5,-c12)      # front surface
      s4 = rectangle $sur (c2,c10,-c6,-c9)      # outflow surface
      s5 = rectangle $sur (-c3,c10,c7,-c11)     # back surface
      s6 = rectangle $sur (-c4,c11,c8,-c12)     # inflow surface
#
#  volumes
#
   volumes         # See Users Manual Section 2.5

     v1 = brick $vol (s1,s3,s4,s5,s6,s2)
#
#  auxiliary statements
#
   intermediate points  # defines spectral elements
      sidepoints=$nside,subdivision=legendre,midpoints=filled
   plot, eyepoint=(3,3,5) # makes also 3d plot
   norenumber # renumbering is not necessary in this ordered case
end
```

Versions a and b are exactly the same.
The internal spectral elements are defined by the type number 600. Only the coefficients 6, 9, 11, 15 and 16 are unequal to zero and hence must be given. Coefficient 6, 9, 11, and 15 get the value 1, whereas coefficient 16 defines the function f. This function is created by the function subroutine FUNCCF.
At the outer surfaces s1 to s6 essential boundary conditions are prescribed. Since these boundary conditions depend on space function subroutine FUNCBC is used to compute their values. To compare the numerical solution with the exact solution an additional function FUNC is defined which contains the exact solution.
The main program is given by:

```
c ***********************************************************************
c
c      File:  exam8-1-4.f
c
c      Contents:  Main program for the test example described
c                 in the SEPRAN manual standard problems 8-1-4
c                 Artificial analytical example
c                 This program uses the most simple version
c                 Since a function subroutine is used for the solution,
c                 it is not possible to used sepcomp
c
c      Usage:     Compile and link this program with the SEPRAN libraries
```

```
c                 seplink exam8-1-4
c
c                 Run this program with input: exam8-1-4.prb
c
c                 exam8-1-4 < exam8-1-4.prb > exam8-1-4.out
c
c      version 2.0    date   30-12-2003
c
c ***********************************************************************
c
c     --- Main program (trivial)

      program exam814
      call sepcom(0)
      end

c     --- function FUNCBC for essential boundary conditions

      double precision function funcbc(ichois,x,y,z)
      implicit none
      integer ichois
      double precision x,y,z
      funcbc = sin(x)*sin(y)*sin(z)
      end

c     --- function FUNC for exact solution

      double precision function func(ichois,x,y,z)
      implicit none
      integer ichois
      double precision x,y,z

      func = sin(x)*sin(y)*sin(z)
      end

c     --- function FUNCCF for right-hand side

      double precision function funccf(ichois,x,y,z)
      implicit none
      integer ichois
      double precision x,y,z

      funccf = 4.0d0*sin(x)*sin(y)*sin(z)
      end
```

The corresponding input file reads

```
# exam8-1-4.prb
#
#  problem file for program exam8-1-4
#  Artificial analytical example of spectral elements
#  See Manual Exams Section 8.1.4
#
#  To run this file use:
#     sepcomp exam8-1-4.prb
```

```
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   reals
      alpha = 1                 # diffusion parameter
      beta  = 1                 # parameter for linear part
   vector_names
      potential                 # solution of spectral element problem
      exact_solution            # analytical solution
   scalars
      error                     # maximum norm of difference
end
#
#  Define the type of problem to be solved
#
problem                   # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1,(type=600)         # Type number for second order elliptic equation
                                # using spectral elements
                                # See Standard problems Section 8.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      outer_surfaces            # The degrees of freedom at all outer surfaces
                                # are prescribed
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   method = 1, mesh = fem_mesh         # the problem is solved by a finite
                                       # element preconditioner
                                       # A profile storage for the
                                       # preconditioner is used, hence this
                                       # preconditioning matrix is solved
                                       # by an ILU decomposition (direct method)
end

# Create start vector and put the essential boundary conditions into this
# vector
#  See Users Manual Section 3.2.5

essential boundary conditions, sequence_number = 1
   outer_surfaces, (func=1)       # The degrees of freedom at all outer surfaces
                                  # are given by a function (subroutine FUNCBC)
end
```

```
#  Define the coefficients for the problems (first iteration)
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 8.1

coefficients, sequence_number = 1
   elgrp1(nparm=20)     # The coefficients are defined by 20 parameters
      coef6 = $alpha    # alpha_11
      coef9 = coef6     # alpha_22
      coef11= coef6     # alpha_33
      coef15= $beta     # beta
      coef16= (func=1)  # right-hand side is a function defined by
                        # subroutine FUNCCF
end

# Create exact solution for comparison
#  See Users Manual Section 3.2.10

create vector, sequence_number = 1
   func = 1            # exact solution is function defined by subroutine FUNC
end

# input for linear solver
#  See Users Manual Section 3.2.8

solve
   spectral accuracy = 1d-8, print_level = 2   # input for the spectral cg loop
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  See Users Manual Section 3.2.3

structure

   # Create the exact solution
      create_vector %exact_solution, sequence_number = 1

   # fill essential boundary conditions into solution vector
      prescribe_boundary_conditions, vector = %potential, sequence_number = 1

   # Solve the spectral system by finite element preconditioning
      solve_linear_system, vector = %potential, seq_coef = 1, seq_solve = 1//
         fem_preconditioning

   # Compare exact solution with numberical solution by subtracting
   # and computing the max norm of the difference
   # print the norm
      compute_scalar %error, norm_dif = 3, vector 1 = %potential//
         vector 2 = %exact_solution
      print %error
end
end_of_sepran_input
```

Version a is almost identical. See the source that you can get by sepgetex.

Version b, however, has a different main program.
In the main program we use a finite element mesh, i.e. a mesh consisting of trilinear finite elements based upon the nodes of the original mesh, and a spectral mesh as defined by sepmesh. The finite element mesh is created by subroutine `femesh` and is denoted by `kmeshel`. The original mesh is called `kmeshsp`. First the problem is solved at the finite element mesh using a direct solver. The solution is stored in array isolel. This solution is used as initial estimate for the final solution.
Next the right-hand side for the spectral problem is build, not the matrix. The boundary conditions are extracted from array isolel.
The system of equations corresponding to the spectral mesh is not actually build. Instead an iterative procedure is used, which requires the computation of the residual in each step. This process is much faster and requires far less memory than building the complete spectral matrix. Subroutine `pcgrad` is used for this iterative solution. The result is stored in array isol.
See the source that you can get by sepgetex. As output the programs reports the maximum norm

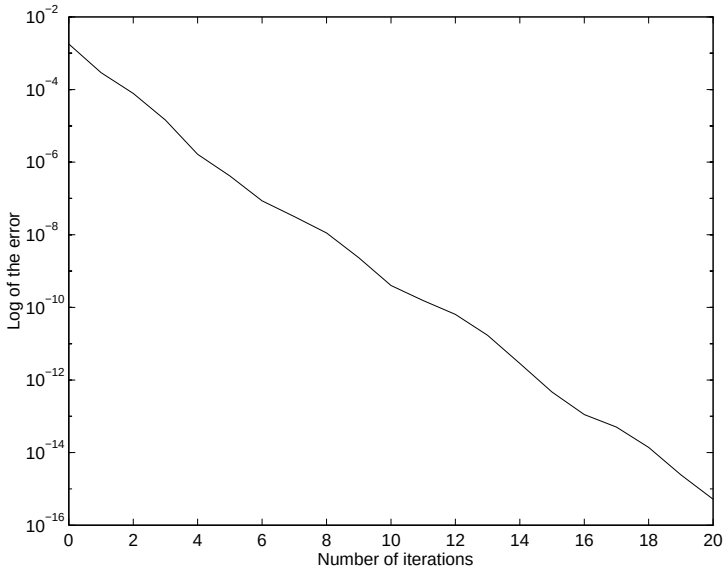


Figure 8.1.4.2: Convergence of the finite element preconditioner; number of iterations versus the log of the residual $\mathbf{Su} - \mathbf{f}$.

of the error:
Error = 4.7E-08

This problem can also be solved using the standard conjugate gradient solver (SOLVEL), with other types of preconditioning. The main advantage of PCGRAD above SOLVEL (for spectral elements) is that no stiffness matrix needs to be build. Each iteration step only a residual needs to be build which can be done very efficiently for the spectral basis functions.

9 Fourth order elliptic and parabolic equations

The following Sections are available:

9.1 The Cahn-Hilliard equation

9.1.1 Example of the solution of the Cahn-Hilliard equation

In this section we consider the solution of the Cahn-Hilliard equation over a simple square domain $\Omega = (0,1)^2$. Further, the initial status of this simplified cell is chosen as a small sinusoidal perturbation around 0.4. Further, we use $f''(c) = \frac{c}{1.3} + \frac{1-c}{0.8} - 4.6c(1-c)$ and $\kappa = 10^{-4}$. Homogeneous natural boundary conditions apply at the boundary of Ω.
To get this example into your local directory use:

```
sepgetex cahn2d
```

and to run it use:

```
sepmesh cahn2d.msh
seplink cahn2d
cahn2d < cahn2d.prb
seppost cahn2d.pst
```

The time-dependent equations to be solved can be found in the manual Standard Problems, Section 9.1.6.
Since the second equation $\Delta c = u$ is time-independent, whereas the first equation is time-dependent, we make it time-dependent by adding an extra term $\rho_u \frac{\partial u}{\partial t}$. In order to make this term in the same range as the error due to time-integration we choose ρ_u equal to Δt^2.
So the equations to be solved are:

$$\begin{aligned} \frac{\partial c}{\partial t} - \nabla \cdot \{M\left[f''(c)\nabla c - \kappa \nabla u\right]\} &= 0, \\ \rho_u \frac{\partial u}{\partial t} - \Delta c + u &= 0 \end{aligned} \tag{9.1.1}$$

The diffusion coefficient $f''(c)$ is taken at the previous time level, making the time-discretization semi-implicit.
The main program contains the function subroutines for initial condition and diffusion coefficient:

```
      program cahn2d

!     --- Standard main program

      integer, allocatable, dimension (:) :: ibuffr
      integer pbuffr, error
      parameter ( pbuffr=25000000)
      allocate(ibuffr(pbuffr), stat = error)
      if (error /= 0) then
         ! space for these arrays could not be allocated
         print *, "error: (cahn2d) could not allocate space."
         stop
      end if  ! (error /= 0)
      call sepcombf ( ibuffr, ibuffr, pbuffr )
      end

!     --- Function func is used to define the initial condition of c

      function func ( ichoice, x, y, z )
      implicit none
      double precision func, x, y, z
      integer ichoice
```

```
      include 'SPcommon/consta'

      func = 0.4 + 0.001*(sin(40*pi*x)+sin(40*pi*y))

      end

!     --- Function funcvect is used to define the diffusion term in the
!         c equation, which is a function of c

      subroutine funcvect ( ichoice, ndim, coor, numnodes, uold, nuold,
     +                      result, nphys )
      implicit none
      integer ichoice, ndim, numnodes, nuold, nphys, i
      double precision coor(ndim,numnodes), uold(numnodes,nphys,nuold),
     +                 result(numnodes,*)
      double precision c

      if (ichoice == 1) then

!     --- ichoice = 1: the diffusivity is given by f''(c)
!         f''(c) = c/1.3 +(1-c)/0.8=4.6c(1-c)

         do i = 1, numnodes
            c = uold(i,1,1)
            result(i,1) = c/1.3+(1-c)/0.8-4.6*c*(1-c)
         end do

      end if

      end
```

The input file for this program `cahn2d.prb` is given by:

```
# cahn2d.prb
#
#  problem file for 2d Cahn Hilliard Equation
#  See Manual Examples Section 9.1.1
#
#  To run this file use:
#     sepcomp cahn2d.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
set warn off  ! suppress warnings

constants           # See Users Manual Section 1.4
   reals
      kappa_eps   = 0.0001          # diffusion parameter
      rho         = 1               # density times heat capacity
      t0          = 0               # initial time
```

```
      t_end       = 4               # end time
      dt          = 0.01            # time step
      rho_u       = dt^2            # artificial density for u equation
   vector_names
      volume_frac     #  volume fraction and lapacian of this vector
                      #  First component c (volume fraction)
                      #  Second component u (Laplacian)
      func_dif        #  diffusivity for c in c equation
end
#
#  Define the type of problem to be solved
#
problem                    # See Users Manual Section 3.2.2

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=808                # Type number for two coupled second order
                                # elliptic equations
                                # See Standard problems Section 3.6
end

# Define how to create the vectors

#  Initial condition of vector volume_frac

create vector, sequence_number = 1
   degfd1, func = 1      ! c is function of x and y defined by
                         ! 0.4 + 0.001*(sin(40 pi x)+sin(40 pi y))
   degfd2, value = 0     ! u = Delta c
end

#  diffusivity for c in c equation as function of volume_frac

create vector, sequence_number = 2
   type = vector of special structure V1  # only one degree of freedom per point
   old_vector = 1, seq_vectors = volume_frac  # ichoice = 1 in funcvect
end


#  Define the coefficients for Laplacian equation
#  All parameters not mentioned are zero
#  See Users Manual Section 3.2.6 and Standard problems Section 3.1

coefficients, sequence_number = 1
   elgrp1 ( nparm=65 )     # The coefficients are defined by 65 parameters

      #  First equation

       coef6 = old solution func_dif, degree of freedom 1
                                    # diffusivity for u in u equation
       coef9 = coef6                # diffusivity for u in u equation
       coef17 = rho                 # rho for time dependence
       coef36 = -kappa_eps          # diffusivity for v in u equation
       coef39 = coef36              # diffusivity for v in u equation
```

```
      #  Second equation

       coef30 = 1                    # Betha^22 = 1 (-Delta c +u = 0 )
       coef32 = rho_u                # rho for time dependence
       coef51 = 1                    # diffusivity for u in v equation
       coef54 = 1                    # diffusivity for u in v equation
end

time_integration, sequence_number = 1
   tinit = t0                                # initial time
   tend  = t_end                             # end time
   tstep = dt                                # time step
   toutinit = t0                             # initial time for output
   toutend  = t_end                          # end time for output
   toutstep = dt                             # time step for output
   method = euler_implicit                   # Time discretization algorithm
   diagonal_mass_matrix                      # The mass matrix is lumped
   mass_matrix = constant                    # and constant for both problems
end

#  Define the structure of the problem
#  In this part it is described how the problem must be solved
#  This part is superfluous, but if you want to solve a more sophisticated
#  problem this is a good start
#
structure                  # See Users Manual Section 3.2.3

   # create initial vector (at t= 0)
   create_vector volume_frac, sequence_number=1

   start_time_loop

     # create the vector func_dif as function of the volume fraction
     # They are used as coefficients for the equation

      create_vector func_dif, sequence_number = 2

     # Perform one time step in the time integration

      time_integration, sequence_number = 1, vector = volume_frac

     # compute and plot bubbles defined by c >= 0.4

      compute_bubble volume_frac, plot, threshold = 0.4

      output

   end_time_loop

   #  print the vectors

   print volume_frac
   print func_dif

end
```

As extra option we have added the option to compute bubbles defined as the part where the volume fraction exceeds the threshold value 0.4. See the Users Manual Section 3.2.3.4 for a description.
In Figure 9.1.1, the solution is plotted over the domain of computation at normalized times $t = 0.1$, $t = 0.25$, $t = 0.5$, $t = 1$, $t = 5$ and $t = 10$. It can be seen in the Figures 9.1.1 that the solution is smooth, but changes rapidly over the interface between adjacent phases. The interfacial width is proportional to $\sqrt{\kappa}$. Further, the initial configuration is unstable since $f''(0.5) < 0$ and hence perturbations start to grow and lipide droplets start to appear. Here, the particles even merged more, hence the number of droplets decreases, however their total occupied area increases. The number of particles per field of view is plotted as a function of time in Figure 9.1.2. The behavior of the number of particles is determined by nucleation, merging and growth of larger droplets at the expense of the dissolution of small-sized particles. In Figure 9.1.3, we show the particle area fraction per field of view, $\frac{|\Omega_p|}{|\Omega|}$ as a function of time. The number of droplets per area of view is shown in Figure 9.1.2. It can be seen that in the initial stages nucleation takes place with a vast increase of lipide area. After the nucleation phenomenon, growth, dissolution and merging takes over, which makes the increase of area less pronounced. Changing parameters like mobility M, gradient energy κ and threshold concentration determines the rate of the process and the shape of the curve. Furthermore, the equilibria are determined from the constants N_1, N_2 and ω.

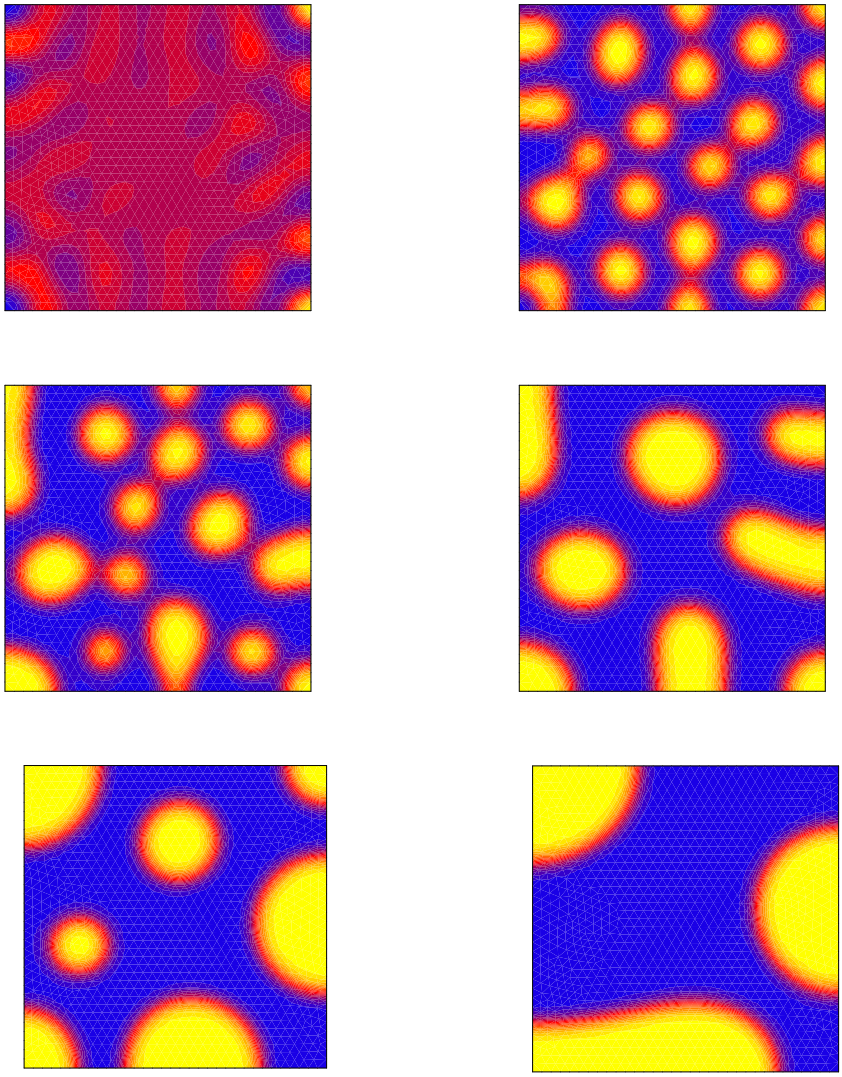


Figure 9.1.1: A contour plot of the solution c at dimensionless times $t = 0.1$ to $t = 10$. One could identify the lipid droplets by the locations where the solution exceeds the value of $\overline{c} = 0.4$.

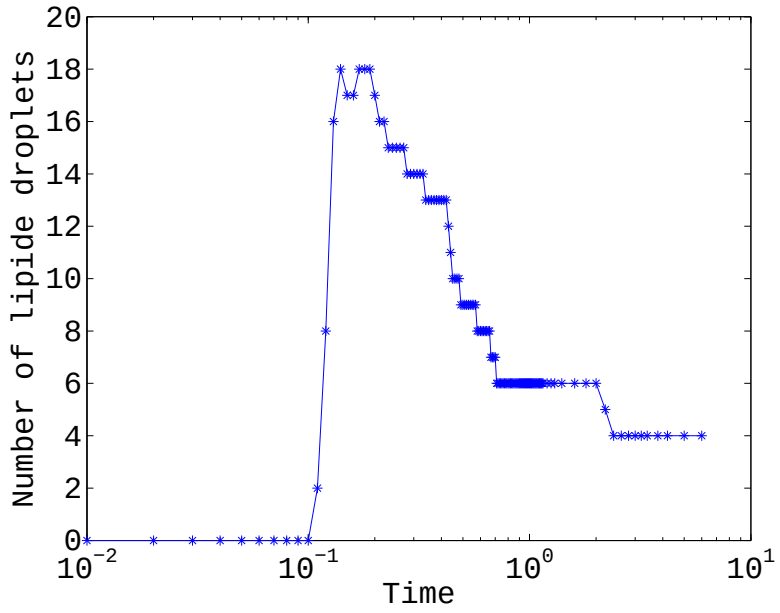


Figure 9.1.2: A plot of the number of lipide droplets per field of view as a function of time.

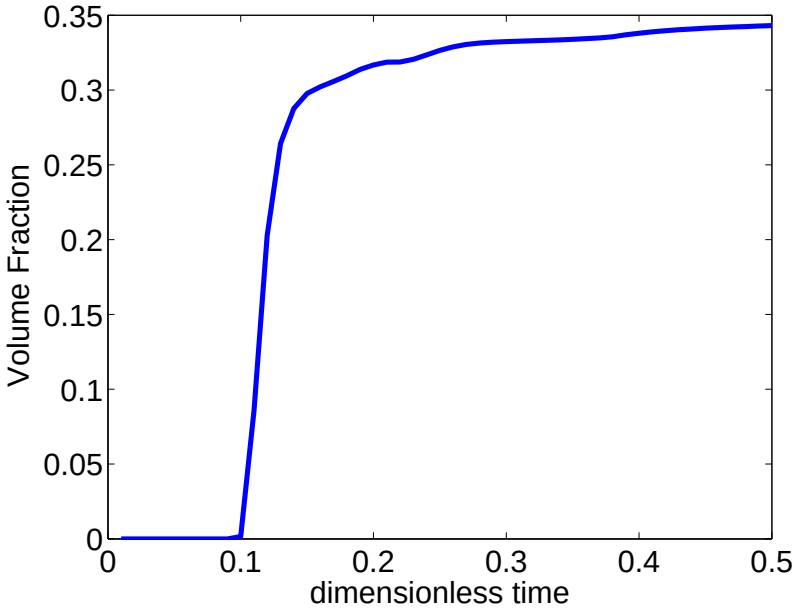


Figure 9.1.3: A plot of a lipid area fraction per field of view as a function of time.

10 Examples of the use of levelset methods

The following Sections are available:

10.1 The dissolution of a particle in a matrix phase

The theory and examples in this Section are based on the work of Etelvina Javierre Perez (2006) and Dennis den Ouden (2012).
In this section we describe the dissolution of a particle within a matrix phase. The interface between particle and matrix phase can have a non-smooth shape. The dissolution of the particle is assumed to be influenced by concentration gradients of a single chemical element within the matrix phase at the particle/matrix boundary and an interface reaction, resulting in a so-called mixed-mode formulation. The mathematical formulation of the dissolution is described by a Stefan problem, in which the location of the interface changes in time. At the interface two boundary conditions are present, one which governs the mass balance at the interface and one that describes the reaction at the interface. Within the matrix phase we assume that the standard diffusion equation applies to the concentration of the considered chemical element.

The mathematical model

Our model is based on the original Stefan problem described by Jožef Stefan in 1890, see Crank (1984). Consider a diffusive phase $\Omega_D(t)$ in which a precipitate $\Omega_P(t)$ has nucleated at some point. Here $\Omega_D(t)$ and $\Omega_P(t)$ are open domains. Let $\Gamma(t)$ denote the interface between the two phases, which represents the moving boundary in our model. Let the concentration c_p within the precipitate $\Omega_P(t)$ be fixed and assume the concentration $c(\boldsymbol{x},t)$ within the diffusive phase $\Omega_D(t)$ to be described by the standard diffusion equation

$$\frac{\partial c}{\partial t}(\boldsymbol{x},t) \;= \nabla\cdot\left(D(\boldsymbol{x},t)\nabla c(\boldsymbol{x},t)\right), \qquad \boldsymbol{x}\in\Omega_D(t), t>0, \tag{10.1.2}$$

where D is the diffusivity of the diffusing chemical element. At the outer boundary of Ω_D, i.e. $\partial\Omega_D(t)\backslash\Gamma(t)$, we assume a no-flux condition, which results into an homogeneous Neumann boundary condition. Furthermore, let $\Omega(t)$ be the open domain defined by

$$\Omega(t) = \left(\Omega_D(t)\cup\Omega_P(t)\right)\setminus\Gamma(t). \tag{10.1.3}$$

At the precipitate/matrix interface $\Gamma(t)$ three physical phenomena occur in sequence during dissolution:

1. Detachment of atoms from the lattice structure of the precipitate phase;
2. Crossing of atoms from within the precipitate into the matrix;
3. Long-range diffusion of atoms into the matrix.

These phenomena occur during growth in the reverse order. In both cases all phenomena put restrictions on the speed at which the interface can move. Many models assume that the diffusive phenomenon is rate-limiting and hence neglect the possible influence of the reaction at the interface given by the first two phenomena. In Vermolen (2007) it has been shown for a plate-like precipitate that the interface reaction can have a significant impact on the dissolution kinetics. Similar to the model in Vermolen (2007) we model the flux of atoms $J_r(\boldsymbol{x},t)$ across the interface by a first-order reaction:

$$J_r(\boldsymbol{x},t) \;=\; K(\boldsymbol{x},t)\left(c_s(\boldsymbol{x},t)-c(\boldsymbol{x},t)\right), \qquad \boldsymbol{x}\in\Gamma(t), t>0. \tag{10.1.4}$$

The flux at the interface within the diffusive phase $\Omega_D(t)$ consists of two parts, the flux $J_m(\boldsymbol{x},t)$ due to movement of the interface itself.

$$J_m(\boldsymbol{x},t) \;=\; c(\boldsymbol{x},t)v_n(\boldsymbol{x},t), \qquad \boldsymbol{x}\in\Gamma(t), t>0, \tag{10.1.5}$$

and the diffusive flux $J_d(\boldsymbol{x},t)$

$$J_d(\boldsymbol{x},t) \;=\; D(\boldsymbol{x},t)\frac{\partial c}{\partial n}(\boldsymbol{x},t), \qquad \boldsymbol{x}\in\Gamma(t), t>0. \tag{10.1.6}$$

In these definitions $K(\boldsymbol{x},t)$ is the interface-reaction speed, $c_s(\boldsymbol{x},t)$ the local equilibrium concentration and $v_n(\boldsymbol{x},t)$ denotes the speed of the interface in the outward normal direction $\boldsymbol{n}(\boldsymbol{x},t)$ from the domain $\Omega_D(t)$ at $\Gamma(t)$. Combining (10.1.4), (10.1.5) and (10.1.6), we arrive at the flux boundary condition

$$K(\boldsymbol{x},t)\left(c_s(\boldsymbol{x},t)-c(\boldsymbol{x},t)\right) \;=\; D(\boldsymbol{x},t)\frac{\partial c}{\partial n}(\boldsymbol{x},t)+c(\boldsymbol{x},t)v_n(\boldsymbol{x},t), \qquad \boldsymbol{x}\in\Gamma(t), t>0. \tag{10.1.7}$$

As we have introduced a new unknown, the interface velocity $v_n(\boldsymbol{x},t)$, we must complete our definition by another boundary condition on $\Gamma(t)$. Using a mass balance on a growing/dissolving precipitate, we arrive at the familiar Stefan condition

$$c_p v_n(\boldsymbol{x},t) \;=\; D\frac{\partial c}{\partial n}(\boldsymbol{x},t)+c(\boldsymbol{x},t)v_n(\boldsymbol{x},t), \qquad \boldsymbol{x}\in\Gamma(t), t>0. \tag{10.1.8}$$

By subtracting (10.1.8) from (10.1.7) we see that the interface velocity $v_n(\boldsymbol{x},t)$ is given by

$$v_n(\boldsymbol{x},t) \;=\; \frac{K(\boldsymbol{x},t)}{c_p}\left(c_s(\boldsymbol{x},t)-c(\boldsymbol{x},t)\right), \qquad \boldsymbol{x}\in\Gamma(t), t>0, \tag{10.1.9}$$

Substituting the above result in either (10.1.7) or (10.1.8), yields that the normal diffusive flux at the interface is given by

$$D(\boldsymbol{x},t)\frac{\partial c}{\partial n}(\boldsymbol{x},t) \;=\; \frac{K(\boldsymbol{x},t)}{c_p}\left(c_s(\boldsymbol{x},t)-c(\boldsymbol{x},t)\right)\left(c_p-c(\boldsymbol{x},t)\right), \qquad \boldsymbol{x}\in\Gamma(t), t>0. \tag{10.1.10}$$

From (10.1.9) we see that the determination of the interface velocity does not involve computing the normal diffusive fluxes at the interface, as opposed to the model used in for example Javierre (2006). A drawback is the introduction of a nonlinear boundary condition on $\Gamma(t)$ for the diffusion problem, in contrast to the simpler Dirichlet condition

$$c(\boldsymbol{x},t) \;=\; c_s(\boldsymbol{x},t), \qquad \boldsymbol{x}\in\Gamma(t), t>0, \tag{10.1.11}$$

used in for example Javierre (2006). Inspection of (10.1.9) shows that if the value of K is large, we will have fast dissolution/growth of the precipitate, indicating diffusion controlled kinetics, whereas a lower value of K leads to slow dissolution/growth, indicating reaction-controlled kinetics.

We assume that the solubility of the considered element at the precipitate/matrix interface inside the diffusive phase, $c_s(\boldsymbol{x},t)$, is known and modeled using the Gibbs-Thomson effect see Perez (2005) and Porter and Easterling (1992).

$$c_s(\boldsymbol{x},t) = c_s^{\infty}(t)exp\left(\zeta\kappa(\boldsymbol{x},t)\right), \tag{10.1.12}$$

where $c_s^{\infty}(t)$ is the solubility of the considered element, ζ a positive physical factor and $\kappa(\boldsymbol{x},t)$ the sum of the principle curvatures of the interface $\Gamma(t)$. The solubility $c_s^{\infty}(t)$ can be derived from thermodynamic databases such as ThermoCalc, see Andersson et al (2002). The parameter ζ is defined as

$$\zeta = \frac{\gamma V_m}{R_g T}, \tag{10.1.13}$$

with γ the interface energy, V_m the molar volume of the precipitate, R_g the gas constant and T the temperature. For a sphere the derivation of (10.1.12) can be found in Perez (2005), leading to $\kappa = 2/R$ where R is the radius of the sphere. By (10.1.12) the equilibrium concentration c_s^{∞} increases for locally convex interfaces, which have positive curvature, and decreases for locally concave interfaces, which have negative curvature. This amplification/dampening will cause the precipitate to grow/dissolve to the configuration with the lowest overall surface tension, i.e. the total energy of the system will be minimized.

10.1.1 1D example of the dissolution of a small particle using a moving grid method

In this section we solve the dissolution of a small particle in a matrix as described in Section 10.1. We solve the diffusion Equation (10.1.2) in the domain Ω_D. Since the domain is time dependent we use a moving mesh method, which is simple in the one-dimensional case. At the outside boundary we use the natural boundary condition $D\frac{dc}{dn} = 0$, which implies that no information has to be given. At the interface we need two boundary conditions, one of which is needed for the evolution of the interface.
We consider two cases:

dirichlet In this case we use the simple boundary conditions of Etelvina Javierre Perez (2006).
At the interface we assume a prescribed boundary condition $c(x,t) = c_s$.
The interface velocity is defined by

$$v_n = \frac{Ddc}{dn}(c_p - c_s) \tag{10.1.1.1}$$

Neumann In this case the non-linear boundary conditions (10.1.9) and (10.1.10).
Hence:

$$v_n = \frac{K}{c_p}(c_s - c). \tag{10.1.1.2}$$

and the mixed boundary condition

$$\frac{Ddc}{dn} + v_n c = v_n c_p \tag{10.1.1.3}$$

In our example we consider the region (0,1). The particle is positioned at the left-hand side in the region (0,S), where S is the interface and the diffusive part is the domain (S,1). The concentration in the particle is defined by `Cpart` and the initial concentration in Ω_D is called `C0`. The equilibrium concentration is given by `Csol`. The parameters in this example are chosen such that the particle shrinks until an equilibrium is reached.
The time step we use depends on the velocity of the interface and is defined by $\Delta t = \frac{CFL\ h}{v_n}$ where h is the minimum step size in Ω_D and CFL a Courant-Lewy-Friedrichs number. For explicit time integration this number should be less than 1 to guarantee stability. Here we use an implicit scheme, but the main reason for this choice is that the update of the interface never jumps over an element. So the new interface is always either in the same element as the old interface or in a direct neighboring element.
In the first 5 steps we multiply the time step by 0.1 to minimize the effect of the transient.
First we consider the case of mixed boundary conditions.

10.1.1.1 Mixed boundary conditions

To get this example into your local directory use:

```
sepgetex partmovebnd1dneu
```

and to run it use:

```
sepmesh partmovebnd1dneu.msh
seplink partmovebnd1dneu
partmovebnd1dneu < partmovebnd1dneu.prb
seppost partmovebnd1dneu.pst
```

The mesh file is trivial:

```
*   partmovebnd1dneu.msh
*
*   Mesh for dissolution of particle (1D)
*   Moving boundary method
*   Mixed boundary condition at interface
*
*   Run:  sepmesh partmovebnd1dneu.msh
*   Creates file meshoutput
*
constants
   reals
      S = 0.615  ! psotion of interface
   integers
      n = 40     ! number of nodes in particle
end
mesh1d

*  definition of user points

  points
      p1= (S)
      p2= (1)

*  curves defining the surfaces:

   curves
      c1= line1(p1,p2,nelm=n)

end
```

The program partmovebnd1dneu is used to compute the update of the coordinates. The file partmovebnd1dneu.prb is used to define the program and is given by

```
*   partmovebnd1dneu.prb
*
*   Problem file for dissolution of particle (1D)
*   Moving boundary method
*   Mixed boundary condition at interface
*
*   Run:  seplink partmovebnd1dneu
```

```
*         partmovebnd1dneu < partmovebnd1dneu.prb
*   Creates files sepcomp.out and sepplot.xxx
*   Uses file meshoutput
*
set warn off
constants
   reals
      D = 1                       # Diffusion parameter
      c0    = 0.3                 # initial concentration in matrix
      csol = 0.33                 # concentration at interface
      cpart = 0.45                # concentration in particle
      k = 1000                    # parameter in mixed boundary condition
      CFL = 0.25                  # CFL number to define time step
      rho_cp      = 1             # Parameter for time derivative
      t0 = 0                      # initial time
      toutstep = 0.0000001        # Make sure that each tie step is printed
      tend = 5                    # end time

   vector_names
      concentration               # concentration in matrix
      displacement                # displacement of nodes
      mesh_vel                    # mesh velocity
   variables
      dt                          # time step
      vn                          # velocity of interface
      h                           # representative step size in space
      dtinv                       # -1/dt
      mass                        # amount of mass in domain
      icount                      # counter for time steps
      cboun                       # concentration at interface
      xboun                       # position of interface
      mass_orig                   # amount of mass at t=0
      mass_loss                   # mass loss
      Ddcdn                       # D dc/dn
      sigma                       # parameter for natural boundary condition
      g                           # parameter for rhs of natural boundary condition
end
problem
   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=800               # Type number for second order elliptic equation
   natbouncond                 # Natural boundary conditions
      bngrp1 = 801             # Type number 801
   bounelms                    # boundary element at interface
      belm 1 = points p1
end

time_integration  ! solve diffusion equation
   tinit = t0                          ! initial time
   tend = tend                         ! end time
   tstep = dt                          ! time step
   toutinit = t0                       ! initial time for output
   toutend = tend                      ! end time for output
   toutstep = toutstep                 ! time step for output
   method = euler_implicit             ! time integration method
```

```
   diagonal_mass_matrix                ! mass matrix is diagonal
end

coefficients    ! coefficients for diffusion equation
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      coef6  = D             # a11 = D
      coef12 = old_solution mesh_vel      # u = mesh velocity
      coef17 = rho_cp        # rho cp
   bngrp1 ( nparm=15 )     # The coefficients are defined by 15 parameters
      coef6 = sigma          # multiplication of c
      coef7 = g              # rhs
end

coefficients, sequence_number = 2  ! for integral
   elgrp1 ( nparm=10 )     # The coefficients are defined by 20 parameters
      coef4  = 1
end

!  Define the steps to be performed by the program

structure

  ! we store the coordinate of p1 (interface) as function of time

   time_history coor, points (p1)

  ! --- Initial concentration

   create_vector concentration, value = c0   ! start concentration

  ! initial mass and some other quatities

   mass = integral concentration ( icheli=2, seq_coef = 2)
   xboun = extract coor (user_point = 1)
   mass_orig = mass+ xboun*cpart
   print mass_orig
   print xboun
   print 'icount      mass         xboun         cboun           vn              dt'

 ! Time integration

   icount = 0
   start_time_loop

   ! compute time step

      cboun = extract concentration (user_point = 1)  ! c at interface
      vn = k/cpart*(csol-cboun)                        ! velocity of interface

      icount = icount+1
      h = min_area                                   ! smallest step size
      dt = cfl*h/vn                                  ! time step

      if ( icount<5 ) then
```

```
!     --- icount < 5
!         In the first 4 steps we reduce dt in order to deal with the
!         discontinuity at the start

         dt = dt*0.1

      end_if

      if ( time+dt>tend ) then

!     --- icount > 1, check if t<tend else adapt dt

         dt = tend-time

      end_if

    !  Compute displacement of mesh (vn dt)

      create_vector displacement, old_vector = 1
      dtinv = -1/dt
      mesh_vel = dtinv*displacement

     ! adapt coordinates

      coor = coor + displacement
      xboun = extract coor (user_point = 1)

     ! Carry out one time step

      sigma =k/cpart*(cpart-cboun)    ! compute sigma for natural bc
      g = sigma*csol                  ! compute g for natural bc

      time_integration concentration

     ! Compute some quantities for output

      mass = integral concentration ( icheli=2, seq_coef = 2)
      mass = mass+ xboun*cpart
      print icount mass xboun cboun vn dt
      output

   end_time_loop

   mass_loss = mass_orig-mass
   print mass_loss

   plot_time_history coor, colors = 10  !red
   set output none
end
```

The problem part is standard for a diffusion equation with natural boundary conditions. The time integration is carried out with an Euler implicit method with a diagonal mass matrix, which is the most simple and stable choice. The part coefficients is standard as can be found in the manual Standard Problems Section 3.1.
The structure part defines the way the program is carried out.

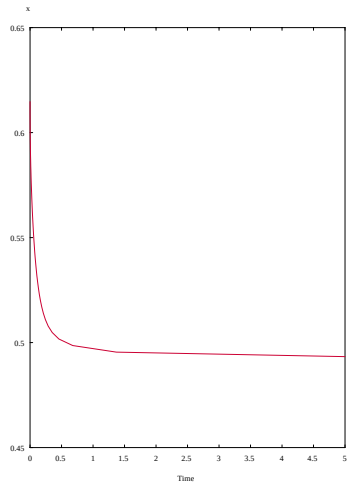


Figure 10.1.1.1: Position of interface as function of time

It starts by defining that we want a time history of the interface node.
Next the initial concentration is set and some output quantities are defined.
In the time loop we compute the time step and compute the displacement of the coordinates in Ω_D by $d_i = -v_n \Delta t \frac{N-i}{N-1}$, where N is the number of nodes in Ω_D.
Since the coordinates of the nodes are changed we need a so-called mesh velocity defined by $v_i = -\frac{x_i^n - x_i^{n-1}}{\Delta t}$, where n defines the time level. The reason is that in the time integration method we approximate $\frac{dc}{dt}$ by $\frac{c^n - c^{n-1}}{\Delta t}$. But since c^n and c^{n-1} are at different positions this is in fact the total derivative $\frac{Dc}{Dt}$, which is equal to $\frac{dc}{dt} + u\frac{dc}{dx}$, where u is minus the mesh velocity $u_i = \frac{x_i^n - x_i^{n-1}}{\Delta t}$. So to get $\frac{dc}{dt}$ we actually need $\frac{Dc}{Dt} - u\frac{dc}{dx}$, hence the minus sign. The mesh velocity is used as one of the coefficients in the convection-diffusion equation.
The non-linear boundary conditions might require a non-linear iteration per time step, but since the time step is small no iteration is necessary and we use the value of v_n as computed from the values at the previous time level.
Figure (10.1.1.1) shows the interface position as function of time.

The main program `partmovebnd1dneu` is only used to compute the displacement as function of the interface displacement.

```
      program partmovebnd1dneu
      call sepcom ( 0 )
      end

      subroutine funcvect ( ichoice, ndim, coor, numnodes, uold, nuold,
     +                      result, nphys )
      implicit none
      integer ichoice, ndim, numnodes, nuold, nphys
      double precision coor(ndim,numnodes), uold(numnodes,nphys,nuold),
     +                 result(numnodes,*)
```

```
      integer i
      double precision vn, dt, fact
      double precision getvar

      select case (ichoice)

      case(1)

!     --- case 1, compute displacement

         vn = getvar ( 'vn' )
         dt = getvar ( 'dt' )
         fact = vn*dt
         do i = 1, numnodes
            result(i,1) = -(numnodes-i)/(numnodes-1d0)*fact
         end do  ! i = 1, numnodes

      case default

!     --- Other values, not programmed
!         Give error and stop

         call errchr('funcvect',1)
         call errsub ( 349, 0, 0, 1)
         call instop

      end select ! case (ichoice)

      end
```

10.1.1.2 Dirichlet boundary conditions

To get this example into your local directory use:

```
sepgetex partmovebnd1ddir
```

and to run it use:

```
sepmesh partmovebnd1ddir.msh
seplink partmovebnd1ddir
partmovebnd1ddir < partmovebnd1ddir.prb
seppost partmovebnd1ddir.pst
```

The case of Dirichlet boundary conditions is slightly different from the mixed case in the sense that we need the derivative the derivative $D\frac{dc}{dn}$ to compute the velocity of the interface. Since $D\frac{dc}{dn}$ at the interface is precisely the flux through the interface it is sufficient to compute the reaction force, which requires an update of the matrix structure and some extra statements in the time integration part.
At the start no reaction force is available so we compute it using derivatives.
The mesh file and main program are exactly the same as for the mixed boundary condition and will not be repeated here. The plot of the interface is also very similar, so we just give the prb file:

```
*   partmovebnd1ddir.prb
*
*   Problem file for dissolution of particle (1D)
*   Moving boundary method
*   Dirichlet boundary condition at interface
*
*   Run:  seplink partmovebnd1ddir
*         partmovebnd1ddir < partmovebnd1ddir.prb
*   Creates files sepcomp.out and sepplot.xxx
*   Uses file meshoutput
*
set warn off
constants
   reals
      D = 1                      # Diffusion parameter
      c0    = 0.3                # initial concentration in matrix
      csol = 0.33                # concentration at interface
      cpart = 0.45               # concentration in particle
      CFL = 0.25                 # CFL number to define time step
      rho_cp      = 1            # Parameter for time derivative
      t0 = 0                     # initial time
      toutstep = 0.0000001       # Make sure that each tie step is printed
      tend = 5                   # end time

   vector_names
      concentration              # concentration in matrix
      displacement               # displacement of nodes
      mesh_vel                   # mesh velocity
      reac                       # reaction_force
   variables
      dt                         # time step
      vn                         # velocity of interface
      h                          # representative step size in space
```

```
      dtinv                        # -1/dt
      mass                         # amount of mass in domain
      icount                       # counter for time steps
      cboun                        # concentration at interface
      xboun                        # position of interface
      mass_orig                    # amount of mass at t=0
      mass_loss                    # mass loss
      Ddcdn                        # D dc/dn
end
problem
   types                          # Define types of elements,
                                  # See Users Manual Section 3.2.2
      elgrp1=800                  # Type number for second order elliptic equation
    essbouncond                   # essential boundary condition at interface (p1)
       points p1
end

time_integration  ! solve diffusion equation
   tinit = t0                           ! initial time
   tend = tend                          ! end time
   tstep = dt                           ! time step
   toutinit = t0                        ! initial time for output
   toutend = tend                       ! end time for output
   toutstep = toutstep                  ! time step for output
   method = euler_implicit              ! time integration method
   diagonal_mass_matrix                 ! mass matrix is diagonal
   boundary_conditions = initial_field ! boundary conditions are stored in solution
   equation 1
      local_options
         reaction_force = reac          ! the reaction force is computed
end

coefficients    ! coefficients for diffusion equation
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      coef6  = D                        # a11 = D
      coef12 = old_solution mesh_vel    # u = mesh velocity
      coef17 = rho_cp                   # rho cp
end

coefficients, sequence_number = 2  ! to compute integral
   elgrp1 ( nparm=10 )     # The coefficients are defined by 10 parameters
      coef4  = 1
end

!  Define the steps to be performed by the program

structure

  ! we store the coordinate of p1 (interface) as function of time

   time_history coor, points (p1)

  ! --- First define the matrix structure
  !     Special i this case is the reaction forces that have to be computed
```

```
   matrix_structure, reaction_forces, storage_method = profile

  ! --- Initial concentration and mesh velocity

   create_vector concentration, value = c0   ! start concentration
   prescribe_boundary_conditions concentration, value = csol, points (p1)

   create_vector mesh_vel, value = 0

  ! The initial reaction force is computed by derivatives
  ! because we have not solved an equation yet

   reac = derivatives ( concentration, seq_coef = 1, type_output = reaction_force &
                        points (p1) )

  ! initial mass and some other quatities

   mass = integral concentration ( icheli=2, seq_coef = 2)
   xboun = extract coor (user_point = 1)
   mass_orig = mass+ xboun*cpart
   print mass_orig
   print xboun
   print 'icount      mass         xboun         cboun           vn              dt'

  ! Time integration

   icount = 0
   start_time_loop

   # compute time step

      cboun = extract concentration (user_point = 1) ! here constant
      Ddcdn = extract reac (user_point = 1)          ! D dc/dn
      vn = Ddcdn/(cpart-csol)                        ! velocity of interface

      icount = icount+1
      h = min_area                                   ! smallest step size
      dt = cfl*h/vn                                  ! time step

      if ( icount<5 ) then

!     --- icount < 5
!         In the first 4 steps we reduce dt in order to deal with the
!         discontinuity at the start

         dt = dt*0.1

      end_if

      if ( time+dt>tend ) then

!     --- icount > 1, check if t<tend else adapt dt

         dt = tend-time
```

```
      end_if

     ! Compute displacement of mesh (vn dt) and mesh velocity

      create_vector displacement, old_vector = 1

      dtinv = -1/dt
      mesh_vel = dtinv*displacement

     ! adapt coordinates

      coor = coor + displacement

     ! Carry out one time step

      time_integration concentration

     ! Compute some quantities for output

      xboun = extract coor (user_point = 1)
      mass = integral concentration ( icheli=2, seq_coef = 2)
      mass = mass+ xboun*cpart

      print icount mass xboun cboun vn dt
      output

   end_time_loop

   mass_loss = mass_orig-mass
   print mass_loss

   plot_time_history coor, colors = 10  !red
   set output none

end
```

10.1.2 1D example of the dissolution of a small particle using a levelset method

This Section treats exactly the same example as in Section (10.1.1). The major difference is that we do not longer use a moving grid method but use a fixed basis mesh in combination with a levelset method. In this particular example there is no gain in using the levelset method but for more complex problems especially in 2d and 3d the profits are large.
The levelset method is based on a fixed background grid. To define the position of the interface as well to distinguish the part were we have a particle and where the matrix phase is, we introduce a levelset function ϕ. ϕ is chosen such that $\phi = 0$ at the interface, $\phi < 0$ in the diffusive phase, and $\phi > 0$ in the particle. Furthermore ϕ must be a distance function (at least in the neighborhood of the interface), which means that $|\phi|$ defines the distance to the interface.
The function ϕ implicitly defines the normal $\boldsymbol{n}$ on the interface by the relation:

$$\boldsymbol{n} = \frac{\nabla\phi}{||\nabla\phi||}. \tag{10.1.2.1}$$

The curvature κ can be computed by:

$$\kappa = -\mathrm{div}\frac{\nabla\phi}{||\nabla\phi||}. \tag{10.1.2.2}$$

These relations are especially important in 2d and 3d.
In this 1d example the function ϕ at start is given by the user. The interface velocity is defined in the same way as in Section (10.1.1). This velocity defines the way we have to update ϕ in a time step due to the movement of the interface. The standard approach is that the new ϕ is the solution of the convection equation:

$$\frac{d\phi}{dt} + w\frac{d\phi}{dx} = 0 \tag{10.1.2.3}$$

where w is an extended velocity field equal to the interface velocity v_n at the interface. In 1d $w = v_n$ is a trivial choice. In order that the relations (10.1.2.1) and (10.1.2.2) remain valid it is necessary that ϕ is a signed distance function, at least in the neighborhood of the interface. This can be achieved for example by solving Equation (3.2.3.4) in the users manual, but our experience is that this is troublesome in complicated situations.
Therefore we compute the "exact" distance for two neighboring rows of elements of the interface provided $|\ ||\nabla\phi|| - 1\ | > \epsilon$, where ϵ is some accuracy. This computation is carries out in an efficient way by looking only for the interface in the neighborhood of points that must be updated.

Since $\phi = 0$ defines the new interface we are able to adapt the mesh to the interface. We start with the background mesh and investigate the intersection of the interface $\phi = 0$ with this mesh. Next we construct a new mesh using these intersections. If an intersection point is close to a node the node is moved to the intersection point, otherwise the element that is intersected is split into 2 new elements. The definition of close is the parameter `accuracy_obstacle` which has a default value 0.3. This means that a node is close to an intersection point if the distance is less than 0.3 $\times$ the element size. The advantage of this approach above the classical level set method is that the interface is approximated more accurately and hence boundary conditions can be satisfied more easily.
The algorithm that is applied can be written as:

Create the mesh
Initialize ϕ and the concentration c; set t = 0
Compute a new levelset mesh based on ϕ and interpolate *phi* and c to this mesh.
while $t < tend$ **do**
  Compute $\nabla\phi$ and v_n
  Compute w by extending v_n over the domain
  Compute Δt
  Update ϕ by solving one time step of Equations (10.1.2.3)
  $t := t + \Delta t$
  Map new ϕ and c to the background mesh
  Compute a new levelset mesh from the background mesh based on ϕ and interpolate ϕ and c to this mesh.
  Compute the mesh velocity
  Update c by solving one time step of the convection-diffusion equation
  Make ϕ a distance function
end while

The mapping of ϕ and c to the background mesh makes the present levelset mesh superfluous. We might remove the level set mesh but to compute the mesh velocity it is necessary to used the value of ϕ at the previous level set mesh. So in fact we are dealing with 3 meshes:

- the background grid
- the present levelset mesh
- the previous levelset mesh

The background grid gets sequence number 1, the active levelset grid number 2 and the new levelset grid number 3. by changing the sequence numbers we reuse the space needed by these grids.
The mesh velocity is defined in the same way as for the moving grid method in Section (10.1.1). The reason that we have to use this velocity is the fact that nodal points in the level set mesh may be moved with respect to the background grid.
Note that the construction of a levelset mesh is much cheaper than constructing a new mesh.

Again we distinguish between the mixed boundary conditions and the Dirichlet boundary conditions.

10.1.2.1 Mixed boundary conditions

To get this example into your local directory use:

```
sepgetex partlevset1dneu
```

and to run it use:

```
sepmesh partlevset1dneu.msh
seplink partlevset1dneu
partlevset1dneu < partlevset1dneu.prb
```

The mesh file for the background grid is trivial:

```
*   partlevset1dneu.msh
*
*   Mesh for dissolution of particle (1D)
*   Levelset method
*   Mixed boundary condition at interface
*
*   Run:  sepmesh partlevset1dneu.msh
*   Creates file meshoutput
*
constants
   reals
      L = 1      ! length of domain
   integers
      n = 80     ! number of nodes in particle
end
mesh1d

*  definition of user points

  points
      p1= (0)
      p2= (L)

*  curves defining the surfaces:

   curves
      c1= line1(p1,p2,nelm=n)
end
```

Note that the number of elements is twice the number used in the moving mesh, because the whole domain is used.
The prb file is given by

```
*   partlevset1dneu.prb
*
*   Problem file for dissolution of particle (1D)
*   Levelset method
*   Mixed boundary condition at interface
*
*   Run:  seplink partlevset1dneu
*         partlevset1dneu < partlevset1dneu.prb
```

```
*   Creates files sepcomp.out and sepplot.xxx
*   Uses file meshoutput
*
set warn off   ! suppress all warnings

#  Define all constants

constants
   reals
      D = 1                       # Diffusion parameter
      c0    = 0.3                 # initial concentration in matrix
      csol = 0.33                 # concentration at interface
      cpart = 0.45                # concentration in particle
      k = 1000                    # parameter in mixed boundary condition
      CFL = 0.25                  # CFL number to define time step
      rho_cp      = 1             # Parameter for time derivative
      t0 = 0                      # initial time
      toutstep = 0.0000001        # Make sure that each tie step is printed
      tend = 5                    # end time
      S0 = 0.615                  # Start value of interface

   vector_names
      concentration               # concentration in matrix
      displacement                # displacement of nodes
      mesh_vel                    # mesh velocity
      phi                         # level set function
      gradphi                     # gradient of phi
      vn                          # velocity of interface extended over domain
      normphi                     # ||gradphi||
      ngradphi                    # gradphi / ||gradphi||
      w                           # vn * ngradphi (pointwise)
      phiold                      # value of phi at start of time step
   variables
      dt                          # time step
      h                           # representative step size in space
      mass                        # amount of mass in domain
      icount                      # counter for time steps
      cboun                       # concentration at interface
      xboun                       # position of interface
      mass_orig                   # amount of mass at t=0
      mass_loss                   # mass loss
      Ddcdn                       # D dc/dn
      sigma                       # parameter for natural boundary condition
      g                           # parameter for rhs of natural boundary condition
      epsdist                     # Distance in which the accuracy of the distance
                                  # function is checked
      nodeb                       # node at zero level set
      maxv                        # maximum velocity
      vnboun                      # velocity of interface
end
general_constants
   accuracy_obstacle = 0.0666666667
end

#  Problem definitions
```

```
problem   # concentration
   num_levelset = 1            # Number of level set functions
   levelset 1, negative_part   # only points with phi_1 < 0 are used
   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=800               # Type number for second order elliptic equation
                               # See Standard problems Section 3.1
   natbouncond                 # Natural boundary conditions
      bngrp1 = 801             # Type number 801
   bounelms                    # boundary element at interface
      belm 1 = zero_levelset 1
problem  2                     # phi

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=800               # Type number for second order elliptic equation
                               # See Standard problems Section 3.1
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   problem 1, storage_method = compact
   problem 2, storage_method = compact
end

#  Coefficients

coefficients, sequence_number = 1, problem = 1 ! coefficients for diffusion equation
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      coef6  = D             # a11 = D
      coef12 = old_solution mesh_vel      # u = mesh velocity
      coef17 = rho_cp        # rho cp
   bngrp1 ( nparm=15 )     # The coefficients are defined by 15 parameters
      coef6 = sigma          # multiplication of c
      coef7 = g              # rhs
end

coefficients, sequence_number = 2, problem = 2  # phi
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      coef12 = old_solution w, degree_of_freedom 1  #  w_1 (velocity)
      coef17 =  1             # rho cp
end

coefficients, sequence_number = 3, problem 1  ! for integral
   elgrp1 ( nparm=10 )     # The coefficients are defined by 20 parameters
      coef4  = 1
end

#  Time integrations

time_integration, sequence_number = 1  ! solve diffusion equation
   reuse_time_parameters               ! the time parameters of phi are reused
```

```
   method = euler_implicit             ! time integration method
   diagonal_mass_matrix                ! mass matrix is diagonal
   seq_coefficients = 1                ! coefficients with seq number 1
end

time_integration, sequence_number = 2   ! integration of phi
   tinit = t0                          ! initial time
   tend = tend                         ! end time
   tstep = dt                          ! time step
   toutinit = t0                       ! initial time for output
   toutend = tend                      ! end time for output
   toutstep = toutstep                 ! time step for output
   method = euler_implicit             ! time integration method
   diagonal_mass_matrix                ! mass matrix is diagonal
   seq_coefficients = 2                ! coefficients with seq number 2
end


!  Define the steps to be performed by the program

structure

  ! we store the coordinate of the interface as function of time

   time_history coor, zero_level_set 1

 # Define ad-hoc level set function phi and c with initial condition

   create_vector phi, func=1, problem = 2   # phi is a function
   create_vector concentration, old_vector = 1, seq_vectors = phi

   h = min_area       ! smallest step size in original mesh
   print h

 # create a new mesh adapted to the zero levelset function
 # This mesh is the standard mesh until a new levelset mesh is created
 # The mesh gets sequence number 2

   make_levelset_mesh, mesh_orig = 1, mesh_subdivide = 2//
      levelset_vector =  phi, interpolate (concentration)

  # Set concentration at interface to c0

   create_vector concentration, zero_levelset 1, value=c0

  # Get some constants

   nodeb = point ( zero_level_set 1)  ! node at interface

   mass = integral concentration ( icheli=2, seq_coef = 3, active_levelset 1)
   xboun = extract coor (node = nodeb)
   mass_orig = mass + xboun*cpart
   print mass_orig
   print xboun
```

```
  print 'icount      mass         xboun         cboun           vn           dt'

# Time integration

  icount = 0
  start_time_loop

     icount = icount+1

     cboun = extract concentration (node = nodeb) ! concentration at interf

     phiold = phi

   # Compute the gradient of phi

     gradphi = derivatives ( phi, icheld = 2 )
     normphi = length ( gradphi )             ! || grad(phi) ||
     ngradphi = gradphi/normphi               ! grad(phi) / || grad(phi) ||

    # compute normal velocity (vector)
     # In the 1d case it is a constant, in 2d we need to solve an equation

     vnboun = K/cpart*(csol-cboun)
     create_vector vn, problem = 2, value = vnboun  ! constant over domain

   # The velocity field for the time-integration of the levelset function
   # is defined by vn n, which is equal to vn grad(phi))

     w = vn*ngradphi

   ! compute time step

     maxv = inf_norm(w)   ! maximum velocity
     dt = cfl*h/maxv

     if ( icount<5 ) then

   ! --- icount < 5
   !     In the first 4 steps we reduce dt in order to deal with the
   !     discontinuity at the start

        dt = dt*0.1

     end_if

     if ( time+dt>tend ) then

   ! --- icount > 1, check if t<tend else adapt dt

        dt = tend-time

     end_if

   # Solve one timestep of the convection equation to compute the new phi
```

```
  time_integration phi, sequence_number = 2

# Next phi and c are interpolated to the basis mesh and a new level mesh
# is created with sequence number 3

  interpolate phi, mesh_in = 2, mesh_out = 1
  interpolate concentration, mesh_in = 2, mesh_out = 1

# Make a new levelset mesh (3) based on phi

  make_levelset_mesh, mesh_orig = 1, mesh_subdivide = 3//
  levelset_vector =  phi, no_interpolation

# Perform one step to compute the new concentration
#  First the concentration is interpolated from mesh 1 to mesh 3
#  The value at the boundary is set to cboun

  interpolate concentration, mesh_in = 1, mesh_out = 3

#  Next it is checked if the new mesh passed a node of the original mesh
#  If so the concentration is copied from mesh 2
#  The boundary value of the concentration is substituted
#  Also the mesh velocity is computed

  levelset_mesh_velocity time_step = dt

# Perform one time step to compute the new concentration on mesh number 3

  nodeb = point ( zero_level_set 1)
  xboun = extract coor (node = nodeb)
  cboun = extract concentration (node = nodeb)
  sigma =k/cpart*(cpart-cboun)
  g = sigma*csol

  time_integration concentration, sequence_number = 1

  mass = integral concentration //
    ( icheli=2, seq_coef = 3, active_levelset 1)
  mass = mass+ xboun*cpart

# Final step: make phi a distance function

  print icount mass xboun cboun vnboun dt
  interchange_mesh ( 2, 3 )   ! interchange meshes 2 and 3 so that
                              ! we never have more than 3 meshes
  output

# make phi a distance function

  epsdist = 2.5*h
  make_distance_function phi

end_time_loop

mass_loss = mass_orig-mass
```

```
   print mass_loss

   plot_time_history coor, colors = 10  !red

   set output none  ! suppress superfluous output

end
end_of_sepran_input
```

The main program is used to compute the initial concentration and the initial function ϕ.

```
      program partlevset1dneu
      call sepcom(0)
      end

      subroutine funcvect ( ichoice, ndim, coor, numnodes, uold, nuold,
     +                      result, nphys )
      implicit none
      integer ichoice, ndim, numnodes, nuold, nphys
      double precision coor(ndim,numnodes), uold(numnodes,nphys,nuold),
     +                 result(numnodes,*)
      integer i
      double precision Cpart, C0
      double precision getconst, getvar

      select case (ichoice)

      case(1)

!     --- case 1, compute initial concentration

         Cpart = getconst ('Cpart')
         C0 = getconst ('C0')
         result(1:numnodes,1) = cpart
         do i = 1, numnodes
            if( uold(i,1,1)<=0d0 ) result(i,1) = c0
         end do  ! i = 1, numnodes

      case default

!     --- Other values, not programmed
!         Give error and stop

         call errchr('funcvect',1)
         call errsub ( 349, 0, 0, 1)
         call instop

      end select ! case (ichoice)

      end

      function func ( ifunc, x, y, z )
      implicit none
      integer ifunc
      double precision func, x, y, z
```

```
      double precision S0
      double precision getconst

      select case (ifunc)

      case(1)

!     --- case 1, compute initial phi

         S0 = getconst ('S0')
         func = S0-x

      case default

!     --- Other values, not programmed
!         Give error and stop

         call errchr('func',1)
         call errint ( ifunc, 1 )
         call errsub ( 1930, 1, 0, 1 )
         func = 0d0
         call instop

      end select ! case (ifunc)

      end
```

10.1.2.2 Dirichlet boundary conditions

To get this example into your local directory use:

```
sepgetex partlevset1ddir
```

and to run it use:

```
sepmesh partlevset1ddir.msh
seplink partlevset1ddir
partlevset1ddir < partlevset1ddir.prb
```

The mesh file and the fortran file are the same as for the mixed boundary conditions. The problem file is given by:

```
*   partlevset1ddir.prb
*
*   Problem file for dissolution of particle (1D)
*   Levelset method
*   Dirichlet boundary condition at interface
*
*   Run:  seplink partlevset1ddir
*         partlevset1ddir < partlevset1ddir.prb
*   Creates files sepcomp.out and sepplot.xxx
*   Uses file meshoutput
*
set warn off   ! suppress all warnings

#  Define all constants

constants
   reals
      D = 1                       # Diffusion parameter
      c0    = 0.3                 # initial concentration in matrix
      csol = 0.33                 # concentration at interface
      cpart = 0.45                # concentration in particle
      k = 1000                    # parameter in mixed boundary condition
      CFL = 0.25                  # CFL number to define time step
      rho_cp      = 1             # Parameter for time derivative
      t0 = 0                      # initial time
      toutstep = 0.0000001        # Make sure that each tie step is printed
      tend = 5                    # end time
      S0 = 0.615                  # Start value of interface

   vector_names
      concentration               # concentration in matrix
      displacement                # displacement of nodes
      mesh_vel                    # mesh velocity
      phi                         # level set function
      gradphi                     # gradient of phi
      vn                          # velocity of interface extended over domain
      normphi                     # ||gradphi||
      ngradphi                    # gradphi / ||gradphi||
      w                           # vn * ngradphi (pointwise)
      phiold                      # value of phi at start of time step
```

```
      reac                        # reaction_force
   variables
      dt                          # time step
      h                           # representative step size in space
      mass                        # amount of mass in domain
      icount                      # counter for time steps
      cboun                       # concentration at interface
      xboun                       # position of interface
      mass_orig                   # amount of mass at t=0
      mass_loss                   # mass loss
      Ddcdn                       # D dc/dn
      epsdist                     # Distance in which the accuracy of the distance
                                  # function is checked
      nodeb                       # node at zero level set
      maxv                        # maximum velocity
      vnboun                      # velocity of interface
end
general_constants
   accuracy_obstacle = 0.0666666667
end

#  Problem definitions

problem   # concentration
   num_levelset = 1            # Number of level set functions
   levelset 1, negative_part   # only points with phi_1 < 0 are used
   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=800               # Type number for second order elliptic equation
                               # See Standard problems Section 3.1
   essbouncond                 # essential boundary condition at interface (p1)
      zero_levelset 1
problem  2                     # phi

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=800               # Type number for second order elliptic equation
                               # See Standard problems Section 3.1
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4

matrix
   problem 1, storage_method = compact, reaction_force
   problem 2, storage_method = compact
end

#  Coefficients

coefficients, sequence_number = 1, problem = 1 ! coefficients for diffusion equation
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      coef6  = D             # a11 = D
      coef12 = old_solution mesh_vel      # u = mesh velocity
      coef17 = rho_cp        # rho cp
```

```
end

coefficients, sequence_number = 2, problem = 2  # phi
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      coef12 = old_solution w, degree_of_freedom 1  #  w_1 (velocity)
      coef17 =  1             # rho cp
end

coefficients, sequence_number = 3, problem 1  ! for integral
   elgrp1 ( nparm=10 )     # The coefficients are defined by 20 parameters
      coef4  = 1
end

#  Time integrations

time_integration, sequence_number = 1  ! solve diffusion equation
   reuse_time_parameters               ! the time parameters of phi are reused
   method = euler_implicit             ! time integration method
   diagonal_mass_matrix                ! mass matrix is diagonal
   seq_coefficients = 1                ! coefficients with seq number 1
   boundary_conditions = initial_field ! boundary conditions are stored in
                                       ! solution
   equation 1
      local_options
         reaction_force = reac         ! the reaction force is computed
end

time_integration, sequence_number = 2  ! integration of phi
   tinit = t0                          ! initial time
   tend = tend                         ! end time
   tstep = dt                          ! time step
   toutinit = t0                       ! initial time for output
   toutend = tend                      ! end time for output
   toutstep = toutstep                 ! time step for output
   method = euler_implicit             ! time integration method
   diagonal_mass_matrix                ! mass matrix is diagonal
   seq_coefficients = 2                ! coefficients with seq number 2
end


!  Define the steps to be performed by the program

structure

  ! we store the coordinate of the interface as function of time

   time_history coor, zero_levelset 1

 # Define ad-hoc level set function phi and c with initial condition

   create_vector phi, func=1, problem = 2   # phi is a function
   create_vector concentration, old_vector = 1, seq_vectors = phi

   h = min_area       ! smallest step size in original mesh
   print h
```

```
# create a new mesh adapted to the zero levelset function
# This mesh is the standard mesh until remove levelset mesh is executed

  make_levelset_mesh, mesh_orig = 1, mesh_subdivide = 2//
     levelset_vector =  phi, interpolate (concentration)

 # Set concentration at interface to csol
 # initilize mesh_vel

  create_vector concentration, zero_levelset 1, value=csol
  create_vector mesh_vel, value = 0

 ! The initial reaction force is computed by derivatives
 ! because we have not solved an equation yet

  reac = derivatives ( concentration, seq_coef = 1, type_output = reaction_force &
                       zero_levelset 1 )

 # Get some constants

  nodeb = point ( zero_levelset 1)  ! node at interface

  mass = integral concentration ( icheli=2, seq_coef = 3, active_levelset 1)
  xboun = extract coor (node = nodeb)
  mass_orig = mass + xboun*cpart
  print mass_orig
  print xboun

  print 'icount      mass         xboun         cboun           vn           dt'

# Time integration

  icount = 0
  start_time_loop

     icount = icount+1

     cboun = extract concentration (node = nodeb) ! concentration at interf

     phiold = phi

   # Compute the gradient of phi

     gradphi = derivatives ( phi, icheld = 2 )
     normphi = length ( gradphi )             ! || grad(phi) ||
     ngradphi = gradphi/normphi               ! grad(phi) / || grad(phi) ||

    # compute normal velocity (vector)
     # In the 1d case it is a constant, in 2d we need to solve an equation

     Ddcdn = extract reac (node = nodeb)           ! D dc/dn
     vnboun = Ddcdn/(cpart-csol)                   ! velocity of interface
     create_vector vn, problem = 2, value = vnboun  ! constant over domain
```

```
# The velocity field for the time-integration of the levelset function
# is defined by vn n, which is equal to vn grad(phi))

  w = vn*ngradphi

! compute time step

  maxv = inf_norm(w)   ! maximum velocity
  dt = cfl*h/maxv

  if ( icount<5 ) then

! --- icount < 5
!     In the first 4 steps we reduce dt in order to deal with the
!     discontinuity at the start

     dt = dt*0.1

  end_if

  if ( time+dt>tend ) then

! --- icount > 1, check if t<tend else adapt dt

     dt = tend-time

  end_if

# Solve one timestep of the convection equation to compute the new phi

  time_integration phi, sequence_number = 2

# Next phi and c are interpolated to the basis mesh and a new mesh
# is created

  interpolate phi, mesh_in = 2, mesh_out = 1
  interpolate concentration, mesh_in = 2, mesh_out = 1

# Make a new levelset mesh based on phi

  make_levelset_mesh, mesh_orig = 1, mesh_subdivide = 3//
  levelset_vector =  phi, no_interpolation

# Perform one step to compute the new concentration
#  First the concentration is interpolated from mesh 1 to mesh 3
#  The value at the boundary is set to cboun

  interpolate concentration, mesh_in = 1, mesh_out = 3
  prescribe_boundary_conditions concentration, value = csol, zero_levelset 1

#  Next it is checked if the new mesh passed a node of the original mesh
#  If so the concentration is copied from mesh 2
#  The boundary value of the concentration is substituted
#  Also the mesh velocity is computed
```

```
      levelset_mesh_velocity time_step = dt

    # Perform one time step to compute the new concentration

      nodeb = point ( zero_levelset 1)
      xboun = extract coor (node = nodeb)
      cboun = extract concentration (node = nodeb)

      time_integration concentration, sequence_number = 1

      mass = integral concentration //
        ( icheli=2, seq_coef = 3, active_levelset 1)
      mass = mass+ xboun*cpart

    # Final step: make phi a distance function

      print icount mass xboun cboun vnboun dt
      interchange_mesh ( 2, 3 )   ! interchange meshes 2 and 3
      output

    # make phi a distance function

      epsdist = 2.5*h
      make_distance_function phi

   end_time_loop

   mass_loss = mass_orig-mass
   print mass_loss

   plot_time_history coor, colors = 10  !red

   set output none

end
end_of_sepran_input
```

10.1.3 2D and 3d versions of the examples in Section 10.1.2

This Section treats the 2d and 3d extensions of the examples in Section 10.1.2. This demonstrates the sue of the level set method in 2d and 3d. At this moment the level set method may only be applied to meshes consisting of linear triangles and tetrahedrons. The intersection of the interface in this case is computed by computing the intersection of the edges of the elements. In the same way as in 1d nodes are moved or elements are subdivided in subtriangles or subtetrahedrons based on the edge intersections.

To get these example into your local directory use for the 2d mixed boundary case:

```
sepgetex partlevsetplane2d
```

and to run it use:

```
sepmesh partlevsetplane2d.msh
seplink partlevsetplane2d
partlevsetplane2d < partlevsetplane2d.prb
```

for the 2d Dirichlet case

```
sepgetex partlevsetplane2ddir
```

and to run it use:

```
sepmesh partlevsetplane2ddir.msh
seplink partlevsetplane2ddir
partlevsetplane2ddir < partlevsetplane2ddir.prb
```

For 3d just replace 2d by 3d.
The triangular mesh used is straight forward and will be repeated here.
The various constants used are stored in the files `partlevsetplane2d.const` and `partlevsetplane2ddir.const`.
The first file is given:

```
*   partlevsetplane2d.const
*
*   Constants file for dissolution of particle (2D) (plane)
*   Levelset method
*   Mixed boundary condition at interface
*

#  Define all constants

constants
   integers
      mult = 1               # Multiplication factor for number of nodes
      base = 16              # basis number of nodes
      n = base*mult          # number of nodes in x-direction
      m = base*mult          # number of nodes in y-direction
   reals
      L = 1                  # Length of domain in x-direction
      b = 1                  # Width of domain in y-direction
      D = 1                  # Diffusion parameter
      c0    = 0.3            # initial concentration in matrix
      cpart = 0.45           # concentration in particle
      kappa = 1000           # parameter in mixed boundary condition
```

```
      CFL = 0.25             # CFL number to define time step
      S0 = 0.615             # Start value of interface
      rho_cp      = 1        # Parameter for time derivative
      t0 = 0                 # initial time
      toutstep = 0.0000001   # Make sure that each tie step is printed
      tend = 5               # end time
      eps = 1e-5             # accuracy for linear solver
      csol0 = 0.301          # initial value for csol
      csolinf = 0.33         # final value for csol
      xbounan = 0.51875      # analytical position of interface at t = infinity
end
```

The problem files very much resemble the 1d case except that now we need vectors in some cases where constants were sufficient. The extension of the normal velocity from the interface is of course arbitrary. We have chosen to use Laplace equation with as Dirichlet boundary condition the computed normal velocity at the interface. Due to the fact that the choice has only little influence on the computations it is sufficient to solve this Laplace equation with the standard accuracy of the iterative linear solvers. In the Dirichlet case the reaction force, which is a flux has to be transformed to nodal values by subdividing by a mass matrix along the interface.
The mean x-value of the interface is compared with the analytical solution in order to get an estimate of the error. To prevent problems with the transient we let csol move gradually from c0 to its final value. This is not necessary but suppress wiggles during the computation. Here we will only give the fortran file and the problem for the mixed boundary case.

```
*   partlevsetplane2d.prb
*
*   Problem file for dissolution of particle (2D) (plane)
*   Levelset method
*   Mixed boundary condition at interface
*
*   Run:  seplink partlevsetplane2d
*         partlevsetplane2d < partlevsetplane2d.prb
*   Creates files sepcomp.out and sepplot.xxx
*   Uses file meshoutput
*
set warn off   ! suppress all warnings

include 'partlevsetplane2d.const'  ! include the constants file

#  Define vectors an scalars

constants
   vector_names
      concentration              # concentration in matrix
      mesh_vel                   # mesh velocity
      phi                        # level set function
      gradphi                    # gradient of phi
      vn                         # velocity of interface extended over domain
      normphi                    # ||gradphi||
      ngradphi                   # gradphi / ||gradphi||
      w                          # vn * ngradphi (pointwise)
      phiold                     # value of phi at start of time step
      sigma                      # variable parameter for mixed boundary cond.
      g                          # parameter for rhs of mixed bc
   variables
```

```
      dt                          # time step
      h                           # representative step size in space
      mass                        # amount of mass in domain
      icount                      # counter for time steps
      cboun                       # concentration at interface
      xboun                       # position of interface
      mass_orig                   # amount of mass at t=0
      mass_loss                   # mass loss
      maxv                        # maximum velocity
      vnboun                      # mean velocity of interface
      delta                       # smallest step size in original mesh
      volrest                     # volume of particle
      csol                        # equilibrium concentration at interface
      xboun_err                   # relative error in boundary
      epsdist                     # Distance in which the accuracy of the distance
                                  # function is checked
      stdev                       # standard deviation in boundary
end

general_constants
   accuracy_obstacle = 0.3   ! defines when points are moved
end

#  Problem definitions

problem                         # concentration
   num_levelset = 1             # Number of level set functions
   levelset 1, negative_part    # only points with phi_1 < 0 are used
   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=800                # Type number for second order elliptic equation
                                # See Standard problems Section 3.1
   natbouncond                  # standard natural boundary conditions
      bngrp1 = 801
   bounelms
      belm 1 = zero_levelset 1 # boundary elements at interface

problem  2                      # phi

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=800                # Type number for second order elliptic equation
                                # See Standard problems Section 3.1
problem  3                      # vn

   num_levelset = 1             # Number of level set functions
   levelset 1, all              # all points are used
   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=800                # Type number for second order elliptic equation
                                # See Standard problems Section 3.1
   essbouncond                  # Define where essential boundary conditions are
                                # given (not the value)
                                # See Users Manual Section 3.2.2
      zero_levelset 1           # Essential boundary conditions on boundary of
```

```
                              # level set
end

#  Define the structure of the large matrix always iterative methods
#  See Users Manual Section 3.2.4

matrix
   problem 1, storage_method = compact            ! concentration
   problem 2, storage_method = compact            ! phi
   problem 3, storage_method = compact, symmetric ! vn
end

#  Coefficients

coefficients, sequence_number = 1, problem = 1
         ! coefficients for convectrion diffusion equation for concentration
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      coef6  = D             # a11 = D
      coef9  = coef 6        # a11 = D
      coef12 = old_solution mesh_vel, degfd1      # u = mesh velocity
      coef13 = old_solution mesh_vel, degfd2      # v = mesh velocity
      coef17 = rho_cp        # rho cp
   bngrp1 ( nparm=15 )     # The coefficients are defined by 15 parameters
      coef6 = old_solution sigma  # sigma
      coef7 = old_solution g      # g
end

coefficients, sequence_number = 2, problem = 2  # phi (convection)
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      coef12 = old_solution w, degree_of_freedom 1  #  w_1 (velocity)
      coef13 = old_solution w, degree_of_freedom 2  #  w_2 (velocity)
      coef17 =  1             # rho cp
end

coefficients, sequence_number = 3, problem = 3 ! coefficients for vn
   elgrp1 ( nparm=20 )     # The coefficients are defined by 20 parameters
      coef6  = 1             # a11 = D
      coef9  = coef 6        # a11 = D
end

coefficients, sequence_number = 4, problem 1  ! for integral
   elgrp1 ( nparm=10 )     # The coefficients are defined by 20 parameters
      coef4  = 1
end

#  Time integrations

time_integration, sequence_number = 1  ! solve diffusion equation
   reuse_time_parameters               ! the time parameters of phi are reused
   method = euler_implicit             ! time integration method
   diagonal_mass_matrix                ! mass matrix is diagonal
   seq_coefficients = 1                ! coefficients with seq number 1
   seq_solve = 1                       ! define input for linear solver
end
```

```
time_integration, sequence_number = 2   ! integration of phi
   tinit = t0                           ! initial time
   tend = tend                          ! end time
   tstep = dt                           ! time step
   toutinit = t0                        ! initial time for output
   toutend = tend                       ! end time for output
   toutstep = toutstep                  ! time step for output
   method = euler_implicit              ! time integration method
   diagonal_mass_matrix                 ! mass matrix is diagonal
   seq_coefficients = 2                 ! coefficients with seq number 2
   seq_solve = 2                        ! define input for linear solver
end


#  Define which linear solver must be used and what accuracy is required

solve, sequence_number = 1  ! concentration (must be accurate)
   iteration_method = cg, accuracy = eps, print_level=0, preconditioning = ilu
end
solve, sequence_number = 2  ! phi and Laplace for vn, standard accuracy
   iteration_method = cg, print_level=0, preconditioning = ilu
end

# Define structure of main program

structure

 # Define ad-hoc level set function phi and c with zero values

   create_vector phi, func=1, problem = 2   # phi is a function
   create_vector concentration, old_vector = 1, seq_vectors = phi

   delta = min_area       ! smallest step size in original mesh
   h = sqrt(2*delta)      ! representative step size
   print h

   csol = csol0           ! initial value for csol

 # create a new mesh adapted to the zero levelset function
 # This mesh is the standard mesh until remove levelset mesh is executed

   make_levelset_mesh, mesh_orig = 1, mesh_subdivide = 2//
      levelset_vector =  phi, interpolate (concentration)

  # Set concentration at interface to c0

   create_vector concentration, zero_levelset 1, value=c0

  # Get some constants

   xboun = mean_value x_coor, zero_levelset 1
   cboun = mean_value concentration, zero_levelset 1

   mass = integral concentration ( icheli=2, seq_coef = 4, active_levelset 1)
   volrest = integral concentration ( icheli=7,non_active_levelset 1)
```

```
  mass_orig = mass + volrest*cpart

  print 'icount      mass         xboun         cboun           vn           dt'
  print icount mass_orig xboun cboun maxv dt

# Time integration

  icount = 0
  start_time_loop

     icount = icount+1

   # The old value of phi is stored in phiold
   # This value is used in levelset_mesh_velocity in order to correct
   # the interpolation in case nodes of the mesh cross
   # the old interface

    phiold = phi

   # Compute the gradient of phi

     gradphi = derivatives ( phi, icheld = 2 )
     normphi = length ( gradphi )             ! || grad(phi) ||
     ngradphi = gradphi/normphi               ! grad(phi) / || grad(phi) ||

    # compute normal velocity (vector)
    # This is done by solving a Laplace equation with Dirichlet boundary
    # conditions at the interface
    # There is no need to use a high accuracy

     create_vector vn, old_vector = 2, seq_vectors = (concentration)//
        zero_levelset 1, problem = 3  # vn at interface

     solve_linear_system vn, seq_coef = 3, seq_solve=2  # solution of laplace equation

   # The velocity field for the time-integration of the levelset function
   # is defined by vn n, which is equal to vn grad(phi))

     w = vn*ngradphi

     maxv = inf_norm(w)   ! maximum velocity
     dt = cfl*h/maxv

     if ( icount<n/2+1 ) then

   ! --- icount < 5
   !     In the first 4 steps we reduce dt in order to deal with the
   !     discontinuity at the start

        dt = dt*0.1

     end_if

     if ( time+dt>tend ) then
```

```
! --- icount > 1, check if t<tend else adapt dt

     dt = tend-time

  end_if

# Solve one timestep of the convection equation to compute the new phi

  time_integration phi, sequence_number = 2

# Next phi and c are interpolated to the basis mesh and a new mesh
# is created

  interpolate phi, mesh_in = 2, mesh_out = 1
  interpolate concentration, mesh_in = 2, mesh_out = 1

  make_levelset_mesh, mesh_orig = 1, mesh_subdivide = 3//
  levelset_vector =  phi, no_interpolation

  xboun = mean_value x_coor, zero_levelset 1

# Perform one step to compute the new concentration
#  First the concentration is interpolated from mesh 1 to mesh 3
#  The value at the boundary is set to cboun

  interpolate concentration, mesh_in = 1, mesh_out = 3
  interpolate vn, mesh_in = 2, mesh_out = 3

#  Next it is checked if the new mesh passed a node of the original mesh
#  If so the concentration is copied from mesh 2
#  The boundary value of the concentration is substituted
#  Also the mesh velocity is computed

  levelset_mesh_velocity time_step = dt

# Update csol

  csol = csolinf
  if ( time<0.1 ) then
     csol = csol0+(csolinf-csol0)*time*10
  end_if

# Compute the vectors that define the mixed boundary condition

  create_vector sigma, old_vector = 3, seq_vectors = (concentration)//
     zero_levelset 1  # sigma at interface
  g = csol*sigma

# Perform one time step to compute the new concentration
# using a convection diffusion equation
# The linear solver requires a rather high accuracy

  time_integration concentration, sequence_number = 1

  cboun = mean_value concentration, zero_levelset 1
```

```
      mass = integral concentration //
        ( icheli=2, seq_coef = 4, active_levelset 1)
      volrest = integral concentration ( icheli=7,non_active_levelset 1)
      mass = mass + volrest*cpart
      stdev = standard_deviation x_coor, zero_levelset 1

    # Final step: make phi a distance function

      print icount mass xboun cboun maxv dt stdev
      interchange_mesh ( 2, 3 )   ! interchange meshes 2 and 3

    # make phi a distance function

      epsdist = 2.5*h
      make_distance_function phi

     # To avoid an endless loop we stop if the number of time steps
     # exceeds 1000

      if ( icount>=1000 ) then
         stop
      end_if

   end_time_loop

   mass_loss = (mass_orig-mass)/mass_orig
   print mass_loss
   xboun_err = abs((xboun-xbounan)/xbounan)
   print xboun_err
   stdev = standard_deviation x_coor, zero_levelset 1
   print stdev

   set output none

end
end_of_sepran_input


      program partlevsetplane2d
      call sepcom ( 0 )
      end

      subroutine funcvect ( ichoice, ndim, coor, numnodes, uold, nuold,
     +                      result, nphys )
      implicit none
      integer ichoice, ndim, numnodes, nuold, nphys
      double precision coor(ndim,numnodes), uold(numnodes,nphys,nuold),
     +                 result(numnodes,*)
      integer i
      double precision vn, dt, fact, Cpart, C0, csol, zeta, kappa
      double precision getconst, getvar

      select case (ichoice)

      case(1)
```

```
!     --- case 1, compute concentration

         Cpart = getconst ('Cpart')
         C0 = getconst ('C0')
         result(1:numnodes,1) = cpart
         do i = 1, numnodes
            if( uold(i,1,1)<=0d0 ) result(i,1) = c0
         end do  ! i = 1, numnodes

      case(2)

!     --- case 2, compute velocity vn

         Cpart = getconst ('Cpart')
         kappa = getconst ('kappa')
         csol = getvar ('csol')

         result(:,1) = kappa/cpart*(csol-uold(:,1,1))

      case(3)

!     --- case 3, compute parameter sigma for natural bc

         Cpart = getconst ('Cpart')
         kappa = getconst ('kappa')

         result(:,1) = kappa/cpart*(cpart-uold(:,1,1))

      case default

!     --- Other values, not programmed
!         Give error and stop

         call errchr('funcvect',1)
         call errsub ( 349, 0, 0, 1)
         call instop

      end select ! case (ichoice)

      end

      function func ( ifunc, x, y, z )
      implicit none
      integer ifunc
      double precision func, x, y, z
      double precision S0
      double precision getconst

      select case (ifunc)

      case(1)

!     --- case 1
```

```
         S0 = getconst ('S0')
         func = S0-x

      case default

!     --- Other values, not programmed
!         Give error and stop

         call errchr('func',1)
         call errint ( ifunc, 1 )
         call errsub ( 1930, 1, 0, 1 )
         func = 0d0
         call instop

      end select ! case (ifunc)

      end
```

The 3d case very much resembles the 2d case.

References

J.O. Andersson, T. Helander, L. Höglund, P. Shi & B. Sundman (2002) Thermo-Calc & DICTRA, computational tools for materials science, Calphad, 26, 273–312 (2002)

Bath Klaus-Jürgen (1982) *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Inc, Englewood Cliffs, New Jersey 07632, 1982.

Bertrand F., P.A. Tanguy and F. Thibault (1997) *A three-dimensional fictitious domain method for incompressible fluid flow problems*, Int. J. for Num. Methods in Fluids, Vol. 25, pp. 719-736, 1997

Brooks A.N. and T.J.R. Hughes (1982) *Stream-line upwind/Petrov-Galerkin formulation for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations*, Comput. Methods Appl. Mech. Engrg., 32, pp 199-259, 1982.

Canuto C., M.Y. Hussaini, A. Quarteroni, and T.A. Zang (1988) *Spectral methods in fluid dynamics.* Springer−Verlag, New York, Berlin.

Caswell B. and M. Viriyayuthakorn (1983) *Finite element simulation of die swell for a Maxwell fluid*, J. of Non-Newt. Fluid Mech. Vol 12, 13-29.

Chun C.K. and S.O. Park (2000) *Fixed-grid finite-difference method for phase-change problems*, Numerical Heat Transfer, Part B 38:, pp 59-73, 2000.

J. Crank (1984) Free and moving boundary problems, Clarendon Press, Oxford

Cuvelier C. (1980) *On the numerical solution of a capillary free boundary problem governed by the Navier-Stokes equations*, Lecture Notes in Physics, Vol 141, pp 373-384, (1980). Editor: M. Jean.

Cuvelier C., A. Segal and A.A. van Steenhoven (1986) *Finite element methods and Navier-Stokes equations.* Mathematics and its applications, Reidel publishing company, Dordrecht.

Van Duijn, C.J., L.A. Peletier, R.J. Schotting (1993) *On the Analysis of Brine Transport in Porous Media*, Eur. J. Appl. Math. Vol. 4, p. 271-302.

Duijn, C.J. van, R.A. Wooding, A. van der Ploeg (2000) *Stability Criteria for the Boundary Layer Formed by Throughflow at a Horizontal Surface of a Porous Medium, to appear*

Gielen A. W. J. (1998) *A continuum approach to the mechanics of contracting skeletal muscle.* PhD thesis, Eindhoven University of Technology, 1998.

Girault V. and P.A. Raviart (1979) *Finite element approximation of the Navier-Stokes equations.* Lecture notes in mathematics, 749, Springer Verlag, Berlin.

Glowinski R. and A. Marrocco (1974) *Analyse numerique du champ magnetique d'un alternateur par element finis et sur-relaxation ponctuelle non lineaire.* Computer Methods in applied mechanics and engineering (1974), 3, 55-85

Glowinski Roland, Tsorng-Way Pan and Jacques Periaux (1994a) *A fictitious domain method for Dirichlet problem and applications* , Computer Methods in Applied Mechanics and Engineering, 111, pp 283-303, 1994.

Glowinski Roland, Tsorng-Way Pan and Jacques Periaux (1994b) *A fictitious domain method for external incompressible viscous flow modeled by Navier-Stokes equations*, Computer Methods in Applied Mechanics and Engineering, 112, pp 133-148, 1994.

Glowinski Roland, Tsorng-Way Pan, Todd I Hesla, Daniel D. Joseph and Jacques Periaux (1999) *A distributed Lagrange multiplier/fictitious domain method for flows around moving rigid bodies: application to particulate flow*, Int. J. for Num. Methods in Fluids, Vol. 30, pp. 1043-1066, 1999

Harrison W.J. (1913) *The hydrodynamical theory of lubrication with special reference to air as a lubricant,* Trans. Cambridge Philos. Soc., (1913), 22, , pp 39-54

Heijningen G.G.J. van and C.G.M. Kassels (1987) *Elastohydrodynamic lubrication of an Oil Pumping Ring Seal.*

Hinze J. O. (1975) *Turbulence,* New York, McGraw-Hill, 2^{nd} edition.

Hirasaki, G.J., J.D. Hellums (1968) *A General Formulation of the Boundary Condition on the Vector Potential in Three-Dimensional Hydrodynamics*, Q. Appl. Math., Vol. 26, p. 331

Hughes Thomas J.R. (1987) *The Finite Element Method. Linear, static and dynamic Finite Element Analysis*, Prentice-Hall, Inc, Englewood Cliffs, New Jersey 07632, 1987.

Hussain A.K.M.F. and W.C. Reynolds (1975) *Measurements in fully developed channel flow*, Journal of Fluids Engineering, pp 568-578.

Javierre Perez Etelvina (2006) *Numerical methods for vector Stefan models of solid-state alloys*, Thesis, Delft University of Technology.

Kruyt N.P., C. Cuvelier and A. Segal (1988) *A total linearization method for solving viscous free boundary flow problems by the finite element method,* Int. J. for Num. Methods in Fluids, Vol. 8, pp. 351-363, 1988

Mizukami A. and T.J.R. Hughes (1985) *A Petrov-Galerkin finite element method for convection-dominated flows: an accurate upwinding technique for satisfying the maximum principle*, Computer Methods in Applied Mechanics and Engineering, 50, pp. 181-193, 1985

Mohr G.A. (1992) *Finite elements for solids, fluids and optimization*, Oxford University Press, Oxford, 1992.

Morgan K.J., J. Périaux and F. Thomasset (1984) *Analysis of Laminar Flow over a Backward Facing Step*, Proceedings of the GAMM Workshop held at Bièvres (Fr.), Vieweg Verlag Braunschweig, 1984.

Ogden R.W. (1984) *Non-linear elastic deformations*, Mathematics and its applications. Ellis Horwood Limited, 1984.

Ouden D den, F.J. Vermolen, L. Zhao, C. Vuik, J. Sietsma (2012) *Application of the level-set method to a diffusion and interface-reaction driven Stefan problem*,

Peng S. H. and W. V. Chang. (1997) *A compressible approach in finite element analysis of rubber-elastic materials.* , Computers & Structures, 62(3):573–593, 1997.

M. Perez (2005) Gibbs-Thomson effects in phase transformations, Scripta Materialia ,52, 709–712 (2005)

Pieters, G.J.M. (2000) *Natural Convection Drive By Groundwater Flow in a Porous Medium*, Master thesis, Delft University of Technology, Faculty of Mathematics.

D.A. Porter & K.E. Easterling (1992) Phase Transformations in Metals and Alloys, 2nd edition, Chapmann & Hall, London (1992)

Rodi W. (1980) *Turbulence models and their application in hydraulics*, Delft, Int. Ass. for Hydraulic Res., 1980.

Segal Guus, Kees Vuik, Kees Kassels (1994) *On the implementation of symmetric and anti-symmetric periodic boundary conditions for incompressible flow*, Int. J. for Num. Methods in Fluids, Vol. 18, pp. 1153-1165, 1994

Souza Neto E.A. de, D.Perić, M.Dutko, and D.R.J. Owen (1996) *Design of simple low order finite elements for large strain analysis of nearly incompressible solids.*, Int. J. Solids Structures, Vol. 33, pp.3277-3296, 1996.

Silliman W.J. and L.E. Scriven (1980) *Separating flow near a static contact line: Slip at Wall and Shape of a Free Surface,* Journal of Computational Physics, Vol. 34, pp. 287-313, 1980

Tanner, R.I, R.E. Nickel and R.W. Bilger (1975) *Finite element methods for the solution of some incompressible non-Newtonian fluid mechanics problems with free surfaces.* Comput. Methods Appl. Mech. Eng., 6, p. 155-174.

Tabata, Masahisa and Kazuhiro Itakura (1995) *Precise computation of drag coefficients of the sphere.* INSAM report no 12 (95-07), Department of Mathematics, Hiroshima University, Higashi-Hiroshima, 739, Japan

Tennekes H. and J. L. Lumley (1974) *An introduction to turbulence,* Cambridge (Mass.), The MIT Press, 3^{rd} printing.

Vahl Davis, G. de (1982) *Natural convection of air in a square cavity: A bench mark numerical solution,* Report 1892/FMT/2, School of Mechanical and Industrial Engineering, University of South Wales.

F.J. Vermolen (2007) On Similarity Solutions and Interface Reactions for a Vector-Valued Stefan Problem, Nonlinear Analysis: Modelling and Control, 12, 269–288 (2007)

van de Vosse F.N. (1987) *Numerical analysis of carotid artery flow.* Thesis Eindhoven University of Technology, 1987.

van de Vosse F.N. and P.D. Minev (1996) *Spectral elements methods : Theory and applications.* EUT Report 96-W-001 ISBN 90-236-0318-5, Eindhoven University of Technology, June 1996.

C. Vuik, A. Segal, J.A. Meijerink (1998) *An efficient preconditioned CG method for the solution of layered problems with extreme contrasts in the coefficients.* Report 98-20. Reports of the Faculty of Math. and Inf., Delft University of Technology. ISSN 0922-5641

Wassenaar R.H. (1994) *Simultaneous heat and mass transfer in a horizontal tube film absorber; numerical tools for present and future absorber designs*, Ph.D. thesis Delft University of Technology, p. 73-75

Wekken B.J.C. van der , R.H. Wassenaar (1988) *Simultaneous heat and mass transfer accompanying absorption in laminar flow over a cooled wall*, Int. J. Refrig. 11, 70-77

Wekken B.J.C. van der, R.H. Wassenaar, A. Segal (1988) *Finite element method solution of simultaneous two-dimensional heat and mass transfer in laminar film flow*, Wärme- und Stoffübertragung (1988) 22, 347-354.

Wooding, R.A. (1960) *Rayleigh Instability of a Thermal Boundary Layer in Flow Through a Porous Medium*, J. Fluid Mech., Vol. 9 p.183-192

Yih S.M. (1986) *Modeling heat and mass transfer in falling liquid films*, in N.P. Cheremisinoff (ed.), Handbook of heat and mass transfer 2, Gulf Publ. Corp., Houston, ch. 5.

Zienkiewicz O.C. and R.L. Taylor (1989) *The Finite Element Method*, Volume 1, Fourth Edition, McGraw-Hill Book Company, London

Zienkiewicz O.C. and R.L. Taylor (1989) *The Finite Element Method*, Volume 2, Fourth Edition, McGraw-Hill Book Company, London

Index