SEPRAN

SEPRA ANALYSIS

USER and TEST EXAMPLES

GUUS SEGAL

USER and TEST EXAMPLES

March 2003

Ingenieursbureau SEPRA
Park Nabij 3
2491 EG Den Haag
The Netherlands
Tel. 31 - 70 3871624
Fax. 31 - 70 3871943

Contents

1 Introduction

In this manual a number of examples of are given, Some of which are provided by users These examples are in first instance meant as illustration of how to make specific applications. However, they also form an excellent starting point for the development of new applications.
Users of SEPRAN are invited to add their own examples to this manual.
Input and source files for all examples treated in this manual can be found in the SEPRAN directory sourceexam/usersexam.

2 Diffusion type equations

In this chapter we treat examples provided by users that are based on diffusion type equations. At this moment the following examples are available:

2.1 An example of a nonlinear time-dependent diffusion equation, with a non-linearity in the time-derivative.

2.1 A nonlinear diffusion problem in time and space

This example has been provided by Fred Vermolen of Delft University of Technology.
To get this example into your local directory give the command:

```
sepgetex nonlindif
```

To run this example use:

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

Outline

The problem has been studied by Pacelli Zitha, Mohamed Darwish and Fred Vermolen. Nonlinear diffusion equations arise in many circumstances. In most cases the nonlinearity comes from a dependency of the diffusion coefficient on the solution itself. In the present case, the nonlinearity is also in the time derivative. The present model is used to compute the migration of contaminants through polymer gels under diffusion and adsorption. The polymer gels can be applied to avoid damage of polluted ground water to the environment. A detailed description of the model and its application to polymer gel barriers is given in Darwish et al. (2001) .

The model equations

The model is based on diffusion and adsorption of the contaminant in the porous medium. We present a scaled model. Let c and u respectively be the *mobile* and *adsorbed* concentration of the contaminant in the aqueous polymer-gel emulsion, and let $D = D(c)$ be the concentration dependent diffusion coefficient of the contaminant, and further we consider a rectangular domain in $\mathbb{R}^2$: $\Omega := \{(x, y) \in \mathbb{R}^2 : 0 < x < 1, 0 < y < 1\}$, then

$$(P_1)\begin{cases} \dfrac{\partial c}{\partial t} = \nabla \cdot (D(c)\nabla c) - \dfrac{\partial u}{\partial t}, & bfx \in \Omega, t > 0, \\ \\ \dfrac{\partial u}{\partial t} = k\,(\psi(c) - u), & bfx \in \Omega, t > 0. \end{cases}$$

The first equation represents the nonlinear diffusion equation. The last term in the right-hand side of the first equation is a sink term due to adsorption. The rate of adsorption is determined from the second equation, where the function $\psi = \psi(c)$ is referred to as an *adsorption isotherm.* The parameter k represents the adsorption rate constant. Physically, this function represents the equilibrium concentration of the adsorbed contaminant in the polymer gel.

Let the domain Ω be divided into two parts: Ω_1 and Ω_2, then we choose our initial condition (IC) such that

$$(IC)\begin{cases} c(bfx, 0) = \begin{cases} 1, & bfx \in \Omega_1, \\ 0, & bfx \in \Omega_2, \end{cases} \\ \\ u(bfx, 0) = \psi(c), \qquad bfx \in \Omega. \end{cases}$$

In other words the initial condition is discontinuous. Over the boundary of the domain, $\partial\Omega$, we apply a homogeneous Neumann boundary condition (BC), that is

$$(BC) \qquad \frac{\partial c}{\partial n} = 0, \qquad bfx \in \partial\Omega, t > 0,$$

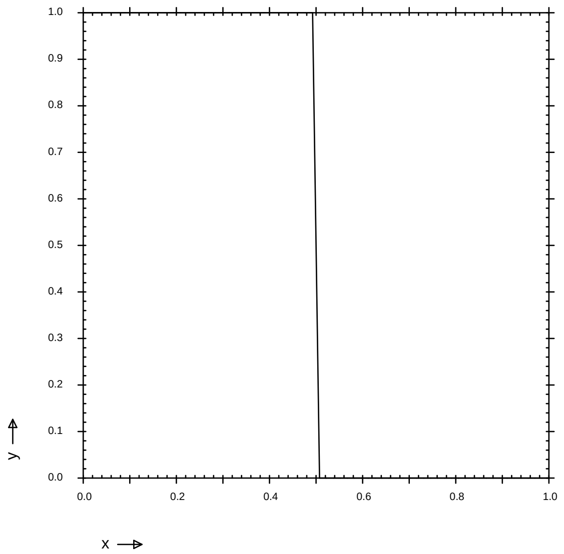

Figure 2.1.1: Horizontal intersection of the initial profile of the concentration.

where **n** represents the outward normal vector on $\partial\Omega$. Note that the function u in (P_1) is only subject to a differential equation in time and that hence no boundary condition at $\partial\Omega$ for u has to be specified. Our purpose is to illustrate a nonlinearity in the time-derivative of the diffusion equation. To do so, we consider the case of equilibrium adsorption. Let the reaction be infinitely fast, then $k \to \infty$ and hence $u \to \psi(c)$. The equations in Problem (P_1) then change into

$$(P_2) \begin{cases} \dfrac{\partial}{\partial t}(c + \psi(c)) = \nabla \cdot (D(c)\nabla c), & bfx \in \Omega, t > 0, \\ c(bfx, 0) = \begin{cases} 1, & bfx \in \Omega_1, \\ 0, & bfx \in \Omega_2, \end{cases} & \\ \dfrac{\partial c}{\partial n} = 0, & bfx \in \partial\Omega, t > 0. \end{cases}$$

For the present we take $\psi(c) = \dfrac{c^2}{2}$, $D(c) = c$ and hence we also have a nonlinearity in the time. As a consequence the differential equation can be written as:

$$(c+1)\frac{\partial c}{\partial t} = \nabla \cdot (D(c)\nabla c)\,. \tag{2.1.1}$$

The case that $D(c) = c$ is interesting, since for $c = 0$ the diffusion coefficient vanishes and a degenerate diffusion equation results. This degenerate diffusion equation gives rise to the existence of an interface (see for instance Logan (1994)) for $c = 0$. We choose $\Omega_1 = \{(x, y) \in \Omega : 0 < x < 1/2$ and $\Omega_1 = \{(x, y) \in \Omega : 1/2 < x < 1$.
Some results are displayed in Figures 2.1.1, 2.1.2, 2.1.3 and 2.1.4. Figure 2.1.1 displays the intersection of the initial profile. Figure 2.1.2 shows the profile with the free boundary that did not reach the opposite boundary yet. Figure 2.1.3 shows a horizontal intersection for a large time. Finally, we show an example of a contoured plot of the solution in Figure 2.1.4.

Mass conservation

It can be seen from Figures 2.1.1 and 2.1.3 that mass conservation of the mobile concentration c is

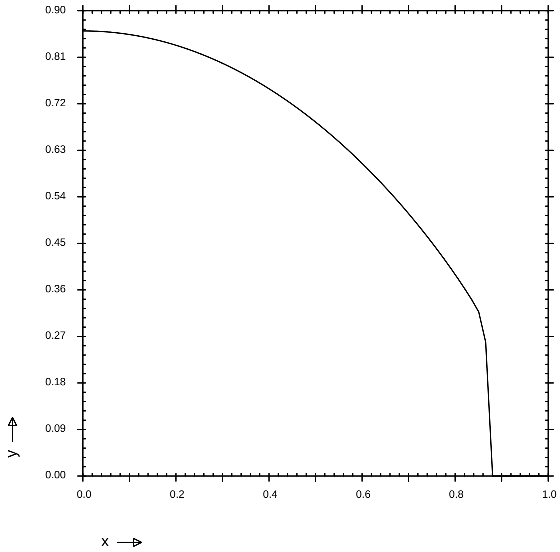

Figure 2.1.2: Horizontal intersection of the profile with a free boundary.

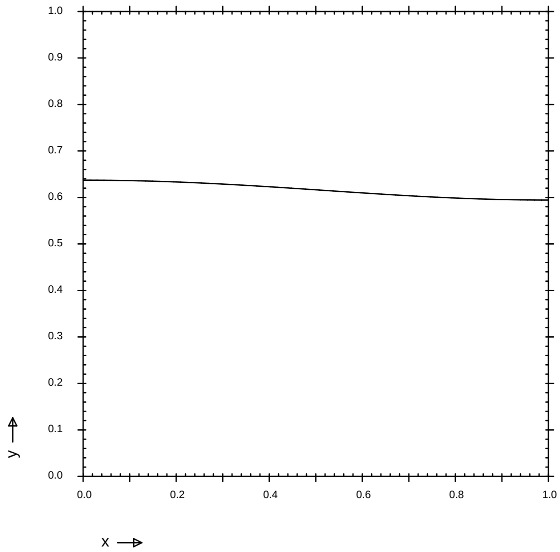

Figure 2.1.3: Horizontal intersection of the profile after the free boundary reaches the opposite boundary.

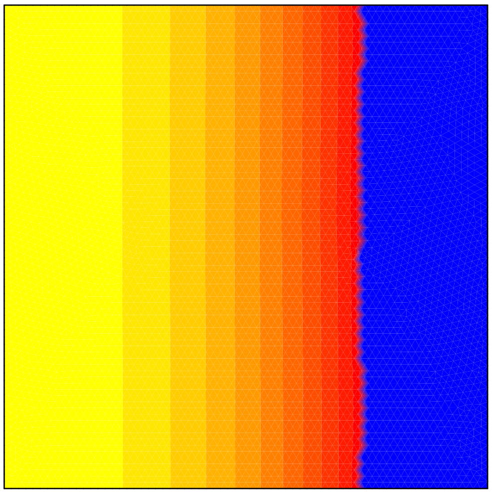

Figure 2.1.4: Coloured contour plot of the solution c.

violated. Physically, this is explained by the fact that some of the concentration c is transformed into adsorbed contaminants u. Hence, the overall contaminant concentration $u + c$ should be conserved. Furthermore, the mobile concentration c should decrease during the process.

Let the function c satisfy (P_2), then one can show that the quantity $\int_\Omega (c + \psi(c))\, d\Omega$ is conserved, i.e. this quantity does not change in time. Furthermore, for this c it can be shown that the quantity $\int_\Omega c\, d\Omega$ respectively decreases for $\psi''(c) < 0$, increases for $\psi''(c) > 0$. This quantity stays constant when $\psi''(c) = 0$, i.e. when $\psi = \psi(c)$ is linear.

In our test problem we have $\psi''(c) > 0$ and hence $\frac{d}{dt} \int_\Omega c\, d\Omega > 0$, which explains the violation of the mass balance in Figures 2.1.1 and 2.1.3.

3 Free surface problems

In this chapter we treat examples provided by users that are dealing with free surfaces. At this moment the following types of examples are available:

3.1 Shape optimization.
This section contains example programs for shape optimization.

3.1 Shape optimization

In this section we treat examples provided by users that are dealing with shape optimization. At this moment the following examples are available:

3.1.1 Adjoint shape optimization for steady free-surface flows.
A steady inviscid free-surface flow problem by means of the adjoint optimal shape design method is solved.

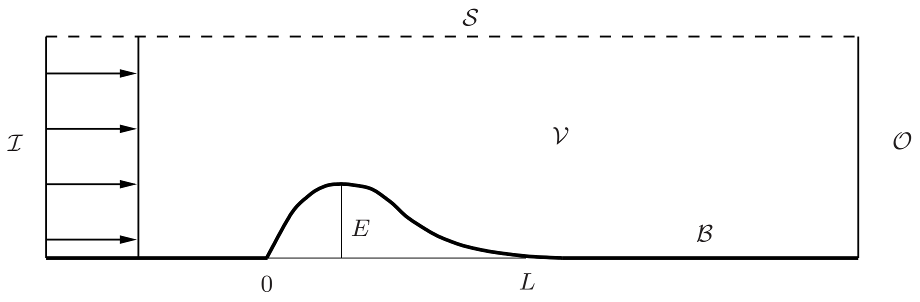

Figure 3.1.1.1: Illustration of the free-surface flow problem

3.1.1 Adjoint shape optimization for steady free-surface flows

This example has been provided by Harald van Brummelen of CWI in Amsterdam.
It concerns a subcritical example and a supercritical one To get these example into your local directory give the command:

```
sepgetex opt_shape_sub
```

or

```
sepgetex opt_shape_super
```

To run these examples use:

```
seplink meshopt_shape_sub
meshopt_shape_sub < opt_shape_sub.msh
view mesh
seplink opt_shape_sub
opt_shape_sub < opt_shape_sub.prb
seppost  opt_shape_sub.pst
view results
```

or

```
seplink meshopt_shape_sup
meshopt_shape_sup < opt_shape_sup.msh
view mesh
seplink opt_shape_sup
opt_shape_sup < opt_shape_sup.prb
seppost  opt_shape_sup.pst
view results
```

In this section we investigate the numerical solution of a steady inviscid free-surface flow problem by means of the adjoint optimal shape design method. The considered problem is illustrated in Figure 3.1.1.1. The method is described in detail in van Brummelen (2001).

Each iteration consists of the following operations:

(A1) For a given position of the free-boundary $\mathcal{S}$, solve the primal problem:

$$\begin{aligned}
\Delta\phi &= 0\,, && \text{in } \mathcal{V}\,,\\
\mathbf{n}\cdot\operatorname{grad}\phi &= 0\,, && \text{on } \mathcal{S} \text{ and } \mathcal{B}\,,\\
\mathbf{n}\cdot\operatorname{grad}\phi &= -1\,, && \text{on } \mathcal{I}\,,\\
\mathbf{n}\cdot\operatorname{grad}\phi &= 1\,, && \text{on } \mathcal{O}\,,
\end{aligned} \tag{3.1.1.1}$$

for ϕ. This system is singular and must therefore be solved iteratively.

(A2) Solve the dual problem for λ:

$$\begin{aligned} \Delta\lambda &= 0\,, && \text{in } \mathcal{V}\,, \\ \mathbf{n}\cdot\operatorname{grad}\lambda &= \mathbf{t}\cdot\operatorname{grad}(p\mathbf{t}\cdot\operatorname{grad}\phi)\,, && \text{on } \mathcal{S}\,, \\ \mathbf{n}\cdot\operatorname{grad}\phi &= 0\,, && \text{on } \mathcal{I},\mathcal{O} \text{ and } \mathcal{B}\,, \end{aligned} \tag{3.1.1.2}$$

with the pressure $p := \frac{1}{2} - (\frac{1}{2}|\operatorname{grad}\phi|^2 + \mathrm{Fr}^{-2}x_2)$.

(A3) Determine the gradient:

$$\mathit{grad} = -\lambda\,\mathbf{nn} : \operatorname{grad}\operatorname{grad}\phi - \sum_{j=1}^{d-1}\mathbf{t}_j\cdot\operatorname{grad}(\lambda\,\mathbf{t}_j\cdot\operatorname{grad}\phi) - \frac{p^2}{2R} - p\,\mathrm{Fr}^{-2}\mathbf{n}\cdot\mathbf{e}_d\,. \tag{3.1.1.3}$$

(A4) Apply a preconditioner to extract the correction, $\beta(\mathbf{x})$, from the gradient. For details see van Brummelen (2001).

(A5) Adjust the boundary $\mathcal{S}$ according to:

$$\mathcal{S} := \{\mathbf{x} + \beta(\mathbf{x})\,\mathbf{n}(\mathbf{x}) : \mathbf{x}\in\mathcal{S}\}\,.\,. \tag{3.1.1.4}$$

This implies that the boundary $\mathcal{S}$ is shifted in its normal direction.

The characteristic parameter in this problem is the Froude number Fr. If $\mathrm{Fr} > 1$, the flow is called supercritical. In that case, the surface elevation takes the form of a single “bump”. If $\mathrm{Fr} < 1$, the flow is called subcritical. The surface elevation then assumes the form of a standing wave, downstream of the obstacle. The structure section in the prb file contains the actual free-surface iteration (A1)–(A5). The operations (A3)–(A5) are contained in the subroutine `funcsolcr` in `opt_shape_sub.f`.

A typical feature of this problem is that the primal and dual boundary value problem are coupled; the right-hand side of Equation 3.1.1.2 depends on the solution of Equation 3.1.1.1. The subroutine `elem2` in `opt_shape_sub.f` provides for the coupling:

In the figures below we have plotted the convergence behavior of the adjoint method for the sub- and supercritical test case. Moreover, the computed surface elevation is compared with measurements. From the figures it appears that the adjoint method converges fast for the supercritical test case and slow for the subcritical test case. The numerical method reproduces the aforementioned typical behavior of free-surface flows for sub- and supercritical flow.

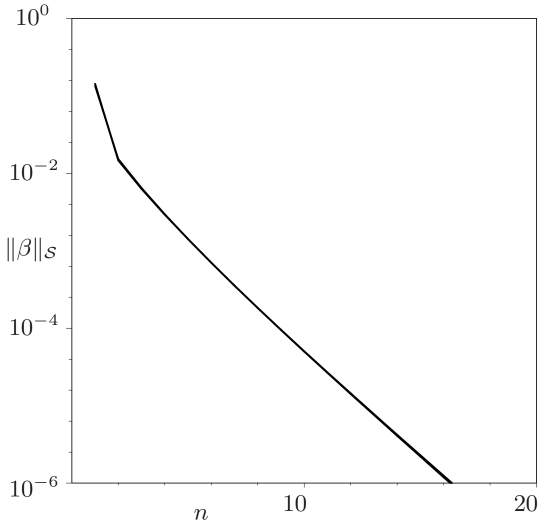

Figure 3.1.1.2: Supercritical test case: convergence ($L/144$ and $L/72$ coincide)

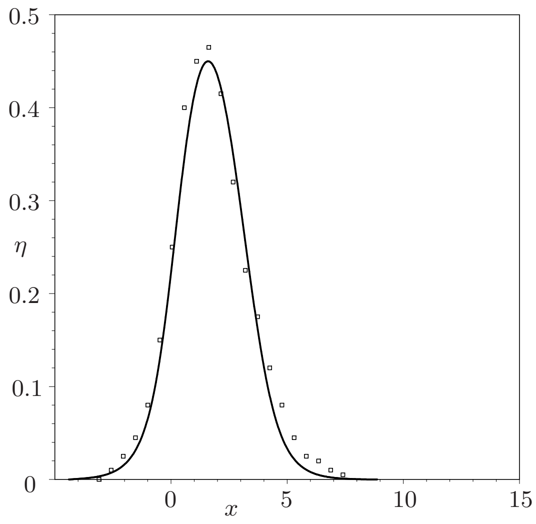

Figure 3.1.1.3: Supercritical test case: computed surface elevation with $h = L/144$ (*solid line*) and measurements from Cahouet (1984) (*markers only*)

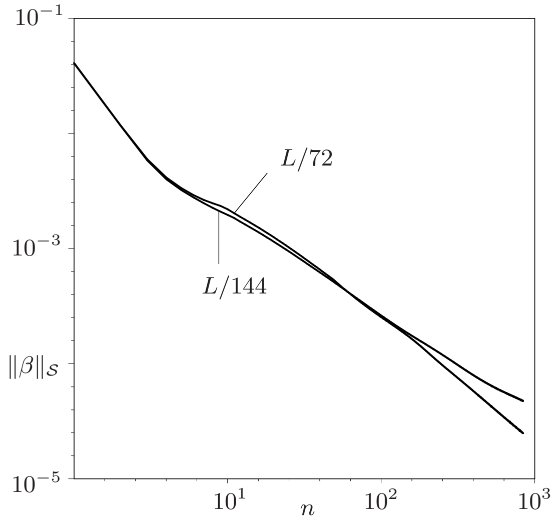

Figure 3.1.1.4: Subcritical test case: convergence

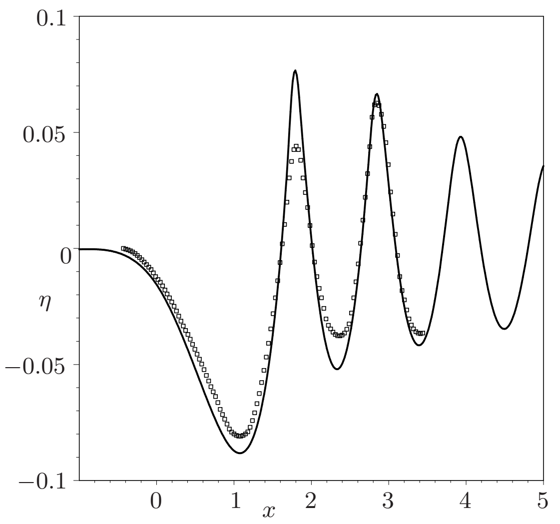

Figure 3.1.1.5: Subcritical test case: computed surface elevation with $h = L/144$ (*solid line*) and measurements from Cahouet (1984) (*markers only*)

4 Flow problems

In this chapter we treat examples provided by users that are dealing with flow problems. At this moment the following types of examples are available:

4.1 Navier Stokes Equations
This section contains example programs for Navier Stokes

4.2 Navier Stokes Equations coupled with heat equation
This section contains example programs for Navier Stokes in combination with the heat equation.

4.1 Navier Stokes Equations

In this section we treat test examples and examples of users that are dealing with the Navier Stokes Equations
At this moment there are the examples available:

4.1.1 The no-flow problem

A standard test problem to check the mass conservation is the so-called no flow problem. Consider a cavity in $x-z$ plane with a bump at the bottom, like the one in Figure 4.1.1.1, which shows also the curves used in the mesh generation. On all walls of the cavity we apply a no-slip condition. Furthermore, the driving force is set equal to ρg, with g the acceleration of gravity and ρ the density. We suppose that the fluid satisfies the incompressible (Navier-) Stokes equations. One immediately verifies that the exact solution of this problem has a zero velocity and a pressure that is equal to $c-\rho g z$, with c an arbitrary constant and z the coordinate in z-direction.
To get this example into your local directory use:

```
sepgetex noflowxx
```

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	bilinear quadrilateral, penalty method
21	6	902	biquadratic quadrilateral, integrated method
31	3	903	linear triangle, Taylor Hood
32	4	903	quadratic, Taylor Hood

To run the problem use

```
sepmesh noflowxx.msh
sepcomp noflowxx.prb
seppost noflowxx.pst
```

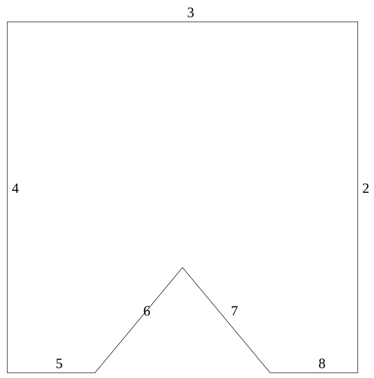

Figure 4.1.1.1: Definition of the curves in the bump

The mesh input file for the case $xx = 11$ is given by

```
#  noflow12.msh
#
#  mesh file for 2d noflow problem
#  See Manual User and Test examples Section 4.1.1
#
#  To run this file use:
#     sepmesh noflow12.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
      l1 = {0.25*$length} # Relative start of bump
      l2 = {0.5*$length}  # Midpoint of bump, also midpoint of bottom
      l3 = {$length-$l1}  # Endpoint of bump
      h1 = 0.3            # Height of bump
   integers
      n1 = 4              # number of elements in length direction along part
                          # of lower wall before bump
      n2 = 4              # number of elements in length direction along one
                          # half of the bump
      n = {2*($n1+$n2)}   # number of elements in length direction
      m = 10           # number of elements in width direction
      shape_cur = 2    # quadratic elements along the curves
      shape_sur = 5    # bilinear quadrilaterals in 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)              # Left under point
      p2=($length,0)        # Right under point
      p3=($length,$width)   # Right upper point
      p4=(0,$width)         # Left upper point
      p5 = ( $l1, 0 )       # Starting point of bump
      p6 = ( $l2, $h1)      # Mid point of bump
      p7 = ( $l3, 0 )       # End point of bump
#
#  curves
#
   curves           # See Users Manual Section 2.3
                    # Quadratic elements are used
      c1=curves(c5,c6,c7,c8)              # lower wall
      c2=line $shape_cur (p2,p3,nelm=$m)   # right-hand side boundary
      c3=line $shape_cur (p3,p4,nelm=$n)   # upper wall
      c4=line $shape_cur (p4,p1,nelm=$m)   # leftt-hand side boundary
      c5=line $shape_cur (p1,p5,nelm=$n1)  # left-hand side part of lower wall
      c6=line $shape_cur (p5,p6,nelm=$n2)  # left-hand side part of bump
      c7=line $shape_cur (p6,p7,nelm=$n1)  # right-hand side part of bump
      c8=line $shape_cur (p7,p2,nelm=$n2)  # right-hand side part of lower wall
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
```

```
                    # Quadratic triangles are used
      s1=rectangle $shape_sur (c1,c2,c3,c4)

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

end
```

Figure 4.1.1.2 shows the mesh created. The corresponding problem input file reads:

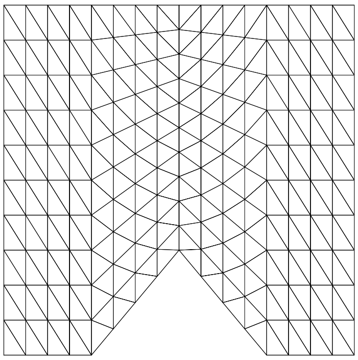

Figure 4.1.1.2: Mesh for no-flow problem

```
# noflow12.prb
#
#  problem file for 2d noflow problem
#  penalty function approach
#  problem is stationary and non-linear
#  See Manual User and Test examples Section 4.1.1
#
#  To run this file use:
#     sepcomp noflow12.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
      g         = 9.81               # gravity
   vector_names
      1: velocity
      2: 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(c1 to c4)           # No-slip walls
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 (vector 1)
   prescribe_boundary_conditions, vector = %velocity
   solve_nonlinear_system, vector = %velocity
 # Compute the pressure (vector 2)
   derivatives, seq_coef=1, seq_deriv=1, vector = %pressure
 # Write the results to a file
   output
end

#  Define the structure of the large matrix
#  See Users Manual Section 3.2.4
matrix
   method = 2                    # 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
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          # 9:  Gravity force (y direction)
      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

# 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

#  Write both velocity and pressure to output file for postprocessing
#  See Users Manual, Section 3.2.13

output
   write 2 solutions
end

end_of_sepran_input
```

In all cases the velocity is (almost zero) and the pressure is linear as expected. However, in the case (12) of the bilinear quadrilateral, which is an element not satisfying the Brezzi Babuska condition, we have a non-zero flow and a pressure which is not exactly linear as shown in Figures 4.1.1.3 and 4.1.1.4. Since in this case the continuity equation is not satisfied exactly an artificial velocity field arises.

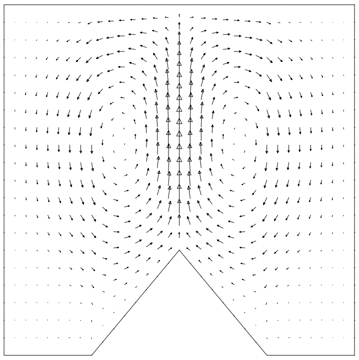

Figure 4.1.1.3: Velocity for no-flow problem (bilinear quad)

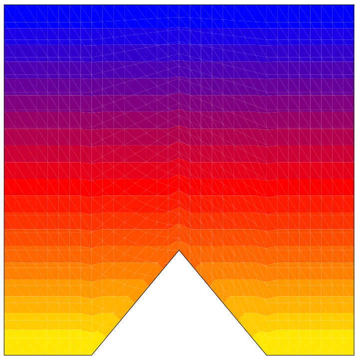

Figure 4.1.1.4: Pressure for no-flow problem (bilinear quad)

4.2 Navier Stokes Equations coupled with heat equation

In this section we treat examples provided by users that are dealing with the Navier Stokes Equations in combination with the heat equation.
At this moment the following examples are available:

4.2.1 Developing plume in the Earth's mantle
The time-dependent problem of a developing plume in the Earth's mantle is solved as an example to illustrate the SEPRAN solution method for a coupled Stokes-heat equation problem.

4.2.1 Rising plume in the Earth's mantle: 2-D and 3-D time-dependent solution of a coupled Stokes-heat equation problem with direct and iterative solvers

This example has been provided by Jeroen van Hunen of the university of Utrecht, faculty of Earth Sciences.
It deals with the time-dependent problem of a developing plume in the Earth's mantle as an example to illustrate the SEPRAN solution method for a coupled Stokes-heat equation problem. We elaborate the example for axi-symmetric 2-D and full 3-D approach, using both direct and iterative solvers.
To get these example into your local directory give the command:

```
sepgetex plumexx_yyy
```

with `xx` equal to 2d or 3d and
`yyy` equal to `dir` (2d only) or `iter`.
dir refers to the direct linear solver and iter to the iterative linear solver.
To run these examples use:

```
sepmesh plumexx_yyy.msh
view mesh
seplink plumexx_yyy
plumexx_yyy < plumexx_yyy.prb
seppost  plumexx_yyy.pst
view results
```

The Earth's mantle can be regarded as a highly viscous fluid, which is (partly) heated from below. The development of a thermal boundary layer at the bottom of the mantle can give rise to the origin of mantle plumes that rise to the surfaces to form 'hotspots' such as Hawaii. Physically, this phenomenon is reasonably well described by a coupled system of the Stokes equation (SP Equation 7.1.2 and 7.1.4 with with the term $\mathbf{v}\cdot\mathbf{v}$ removed since inertia forces are small compared to viscous forces) and a thermal convection-diffusion equation (SP Equation 3.1.1). The non-linear coupling is formed by flow-dependent advection of heat and thermal instabilities from temperature-dependent density variations. These equations are non-dimensionalised with $\mathbf{x} = \mathbf{x'h}$, $t = t'h^2/\kappa$, $\mathbf{v} = \mathbf{v'}\kappa/\mathbf{h}$, and $T = T'\Delta T + T_0$ to:

$$\nabla\cdot\mathbf{v} = \mathbf{0} \tag{4.2.1.1}$$

$$\nabla\cdot(\eta(\nabla v + \nabla v^T)) - \nabla P = RaT \tag{4.2.1.2}$$

$$\frac{\partial T}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{T} - \nabla^{\mathbf{2}}\mathbf{T} = \mathbf{0} \tag{4.2.1.3}$$

where accents are removed for clarity, and the Rayleigh number $Ra = 10^5$ is defined as $Ra = \frac{d\rho/dT\Delta Th^3}{\eta_0\kappa}$, $\kappa = \alpha/\rho c_p$ is the thermal diffusivity, h is the height of the model domain, ΔT the constant maximum vertical temperature difference over the model domain, and $T_0 = 273K$. A simple constitutive equation for temperature-dependent Newtonian rheology defines the viscosity in the Earth's mantle:

$$\eta/\eta_0 = \mathrm{e}^{(}-T\ln\Delta\eta) \tag{4.2.1.4}$$

In this way, material with $T = 0$ is $\Delta\eta$ times more viscous than material with $T = 1$.

We use a vertical cylindric model domain with the boundary conditions as given in Figure 4.2.1.1. Initially (at dimensionless time $t = 0$), the temperature field T is zero in the whole model domain, except for the lower boundary, where $T = \max(0, 1-2\mathrm{r})$ is prescribed to 'feed' the thermal plume. In 2-D, we use axi-symmetry to solve the problem, while in 3-D, we solve for the plume development in one quarter of the full cylinder.

A 2-D model with direct and iterative solution methods

In two dimensions, the radial and vertical coordinates are described by the sepran coordinates x and y. A simple square mesh is defined, using 9-point isoparametric quadrilateral elements of shape number 6 (UM Table 2.2.1) that can be handled by both the direct solution method (penalty function method) as the iterative solvers.
The input file for the mesh generator is the same for the direct solution method as for the iterative solution method.

```
#  plume2d_dir.msh
#
#  mesh file for 2d developing plume problem in the Earth's mantle
#  A direct linear solver is used
#  See the Examples Manual Section 4.2.1
#
#  To run this file use:
#     sepmesh plume2d_dir.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants          # See Users Manual Section 1.4
   integers
      nelm = 20       # Number of elements in horizontal and vertical directions
      cur_shape = 2   # shape of curve elements (quadratic)
      sur_shape = 6   # shape of surface elements (bi-quadratic quadrilaterals)
   reals
      length = 1      # Length of region
      width = 1       # Width of region
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 $cur_shape (p1,p2,nelm=$nelm)     # lower wall
      c2=line $cur_shape (p2,p3,nelm=$nelm)     # outflow boundary
      c3=line $cur_shape (p3,p4,nelm=$nelm)     # upper wall
      c4=line $cur_shape (p4,p1,nelm=$nelm)     # inflow boundary
#
#  surfaces
#
```

```
   surfaces         # See Users Manual Section 2.4
                    # The surface generator quadrilateral is used
                    # The same result is created by rectangle
      s1 = quadrilateral $sur_shape (c1,c2,c3,c4)

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

end
```

A functional description of the boundary conditions and viscosity coefficients make recompilation of the sepcomp standard program necessary:

```
      program plume2d_dir

c     --- Main program for 2d  developing plume problem in the Earth's mantle
c         A direct linear solver is used
c         penalty function approach
c         The heat equation and Stokes equation are solved in a decoupled way
c         In each time step corresponding to the solution of the heat equation,
c         the stationary Stokes equations are solved
c         The main program is necessary because function subroutines are used

      call sepcom(0)

      end

c     --- Function func is used to define the initial temperature along the
c         lower wall

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

      if ( ichoice.eq.1 ) then

c     --- initial temperature distribution:

         func = max(0d0, 1d0-2*x)

      else

c     --- Other values of ichoice not permitted

         print *,'This value of ichoice (',ichoice,') has not been ',
     +           'implemented in function func'

      end if

      end


c     --- Function funcc3 is used to define the viscosity as function
```

```
c         of the temperature and the viscosity contrast for T=0 and T=1

      function funcc3 ( ifunc, x, y, z, numold, maxunk, uold )
      implicit none
      integer ifunc, numold, maxunk
      double precision funcc3, x, y, z, uold(numold,maxunk)

      integer iold, icount
      double precision deta, getconst

c     --- Make the following variables valid in all calls
c         Their values are kept

      save iold, icount, deta

c     --- Initialize icount to 0

      data icount /0/

      if ( ifunc.eq.1) then

c     --- ifunc = 1, compute viscosity

         if ( icount.eq.0 ) then

c        --- First step, get values of iold and deta
c            Since these parameters are saved, they have to computed only once
c            Get sequence number of temperature in solution vector
c            and store in iold
c            Get deta from the input file

            call prgetname ( 'temperature', iold )
            deta=getconst('deta')

c           --- change icount in order to prevent recomputing these constants

            icount = 1

         end if

c        --- Compute viscosity

         funcc3 = exp(-log(deta) * uold(iold,1) )

      else

c     --- Other values of ichoice not permitted

         print *,'This value of ifunc (',ifunc,') has not been ',
     +           'implemented in function funcc3'

      end if
```

```
      end

c     --- Subroutine userout is used to store the computed maximum
c         velocity into the file vmax.dat for postprocessing purposes

      subroutine userout ( kmesh, kprob, isol, isequence, numvec )
      implicit none
      include 'SPcommon/comcons1'
      include 'SPcommon/cuscons'
      include 'SPcommon/ctimen'

      integer kmesh(*), kprob(*), numvec, isol(5,*), isequence(*)

      double precision getvar

      integer icount, iref
      double precision vmax

c     --- Save icount so that the value is kept for a subsequent call

      save icount

c     --- Initialize icount to 0

      data icount /0/

      iref = 31
      if ( icount.eq.0 ) then

c     --- Open the file vmax.dat in the first step (icount=0)
c         Since icount is saved, this is an allowed construction

         open ( unit=iref, file='vmax.dat' )
         icount = 1

      end if
      vmax = getvar('normvel')
      write(iref,*) t, vmax

      write(*,100)' time=', t,': Vmax=', vmax
100   format(a,f10.5,a,f10.5)

      end
```

A direct solution method (penalty function approach) is used in the following sepcomp input file. Axi-symmetry is defined in coefficient 4 of both equations. The solution is written to file every time step, and also the maximum velocity is calculated. The coupling between the velocity and temperature solutions (Stokes and heat equation solutions) is introduced through the coefficient definition. Temperature variations drive the flow through buoyancy forces (coefficient 10 in the Stokes equation), while the flow field is introduced in the heat advection (coefficients 12 and 13 of the heat equation). The temperature dependent viscosity is calculated in `funcc3`.

```
#  plume2d_dir.prb
#
```

```
#  problem file for 2d developing plume problem in the Earth's mantle
#  A direct linear solver is used
#  penalty function approach
#  The heat equation and Stokes equation are solved in a decoupled way
#  In each time step corresponding to the solution of the heat equation,
#  the stationary Stokes equations are solved
#
#  See the Examples Manual Section 4.2.1
#
#  To run this file use:
#     sepcomp plume2d_dir.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      Stokes = 1                     # Sequence number of Stokes problem
      heat = 2                       # Sequence number of heat problem
   reals
      eps      = 1d-6                # penalty parameter for Stokes
      Ra       = 1d5                 # Rayleigh number
      deta     = 1d3                 # viscosity contrast for T=0 and T=1
      rho      = 1                   # density
      kappa    = 1                   # heat conduction
      rho_cp   = 1                   # rho cp
   vector_names
      1: velocity                    # Stokes solution
      2: temperature                 # solution heat equation
   variables
      1: normvel                     # To store maximum velocity
end
#
#  Define the type of problem to be solved
#
problem $Stokes          # See Users Manual Section 3.2.2
                         # Problem description of the Stokes equation

   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
      degfd2,curves(c1)        # Normal velocity on lower wall prescribed
      curves(c2)               # Fixed right-hand side wall
      degfd2,curves(c3)        # Normal velocity on upper wall prescribed
      degfd1,curves(c4)        # Normal velocity on left-hand side wall
                               # prescribed
```

```
                               # All not prescribed boundary conditions
                               # satisfy corresponding stress is zero
problem $heat           # Problem description of the heat equation

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=800               # Type number for heat 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)               # Temperature given on lower wall
      curves(c2)               # Temperature given on right-hand side wall
      curves(c3)               # Temperature given on upper wall
end

#  Define the structure of the large matrices
#  See Users Manual Section 3.2.4
matrix
   method = 1, problem=$Stokes # symmetrical profile matrix for Stokes
                               # Only Stokes with penalty is symmetrical
                               # So a direct method will be applied
   method = 2, problem=$heat   # Non-symmetrical profile matrix for the heat
                               # equation
                               # So a direct 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 a time-dependent problem is coupled with a
#  time-independent problem and also some extra computations are carried out
#
structure                   # See Users Manual Section 3.2.3
   # Create initial conditions:

      create_vector, vector = %velocity, sequence_number 1  # set v = 0
        # Actually setting u = 0, means that the essential boundary conditions
        # are made equal to 0
        # This statement could also be replaced by:
        # prescribe_boundary_conditions, vector = %velocity

   create_vector, vector = %temperature, sequence_number 2  # T given

   # Compute the velocity in the first step by solving the Stokes equations

      solve_linear_system, seq_coef=1, vector = %velocity

   # Compute the maximal norm of the velocity over all points in the mesh
   # Store in normvel

      compute_scalar%normvel, norm=3, vector1=%velocity

   # Write the initial condition to sepcomp.out
```

```
      output, sequence_number=1

   # Time integration:

   start_time_loop

      # solve 1 timestep for T:

         time_integration, sequence_number = 1, vector=%temperature

      # solve stationary v:

         solve_linear_system, seq_coef=1, problem=$Stokes, vector=%velocity

      # Compute the max norm of the velocity
      # Store in normvel

         compute_scalar%normvel, norm=3, vector1=%velocity

      # Use the own-written subroutine userout to
      # print the computed max norm and to write to the file vmax.dat

         user_output

      # Write the solution vectors to sepcomp.out at times defined by
      # toutinit, toutstep and toutend

         output, sequence_number=1

   end_time_loop

end

#  Create initial vectors for velocity and Temperature
#  This part describes which values are used
#  See Users Manual Section 3.2.10

create vector, sequence_number 1, problem $Stokes
   value = 0    # The velocity vector is set to 0
end
create vector, sequence_number 2, problem $heat
   curves (c1), func = 1     # The Temperature at the lower wall is a function
                             # of the position: T = 1-2x
                             # This is defined in function func
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 and 3.1

coefficients, sequence_number=1, problem = $Stokes    # problem 1: Stokes
   elgrp1 ( nparm = 20)                              # The coefficients are defined
                                                     # by 20 parameters
     icoef2 = 1                                      # constitution eq.
```

```
                                                # Newtonian model
     icoef4 = 1                                 # axi-symmetry
     icoef5 = 0                                 # Stokes' flow (no inertia term)
     coef6  = $eps                              # penalty parameter
     coef7  = $rho                              # density of fluid
     coef10 = old solution %temperature, coef=$Ra    # y-dir. body force
     coef12:  sol_func=1                        # eta is a function of the
                                                # temperature and the viscosity
                                                # contrast deta
                                                # Function FUNCC3 is used
end

# Coefficients for the heat equation

coefficients, sequence_number=2, problem = $heat    # problem 2: heat-eq.
   elgrp1 ( nparm = 20)                         # The coefficients are defined
     icoef3= 3                                  # integration rule
     icoef4= 1                                  # axi-symmetry
     coef6 = $kappa                             # thermal conductivity
     coef9 = $kappa                             # thermal conductivity
     coef12= old solution %velocity degree of freedom 1 # u-velocity
     coef13= old solution %velocity degree of freedom 2 # v-velocity
     coef17= $rho_cp                            # rho cp
end

#  Input for the time integration
#  See Users Manual Section 3.2.15

time_integration, sequence_number = 1
   method                 = euler_implicit    # Default time integration scheme
   tinit                  = 0                 # Initial time
   tend                   = 8d-3              # end time
   tstep                  = 1d-4              # time step
   toutinit               = 0                 # Initial time for output
   toutend                = 8d-3              # end time for output
   toutstep               = 1d-4              # time step for output
   seq_coefficients       = 2                 # Defines which coefficients
                                              # are used for the heat eq.
   boundary_conditions    = initial_field     # The boundary conditions are
                                              # constant and equal to the
                                              # bc's at t=0
   diagonal_mass_matrix                       # The mass matrix is lumped
   print_level            = 2                 # Defines the amount of output
end

#  Write both velocity and temperature to output file for postprocessing
#  See Users Manual, Section 3.2.13
#  Not necessary, because this is the default

output, sequence_number=1
  write 2 solutions
end

end_of_sepran_input
```

As an iterative solution method, the integrated method with standard element 902 is used. In comparison to the direct solution method, as described above, some extra information concerning the iterative solution method must be given. The most stable solution method for both the Stokes equation and the temperature equation turned out to be GMRESR with ILU preconditioner. Renumbering-per-level of the unknowns enables the required ILU factorization of the Stokes equation (Segal & Vuik, (1995)). The matrix is, different from the penalty function method, non-symmetric (Segal & Vuik, (1995)), so that for both the Stokes and the heat equation, matrix `method=6` is used. Apart from the two velocity unknowns, with the integrated method 902, the centroid of the element contains the pressure and its derivatives as extra unknowns. This complicates the calculations of any velocity norm. Therefore, we explicitly picked out the two velocity unknowns to derive the maximum velocity. It also complicates the definition of a convergence criterion of the Stokes solution method: Absolute criteria are difficult to choose, since residues of the pressure, its derivatives, and the velocity components are difficult to compare. Therefore, we used (the default choice of) a relative convergence criterion of the residue as stop criterion for both equations. To obtain an accurate start solution for the velocity field in one step, we use a small convergence criterion for the first solution. Because for all next solutions, the initial residue is already quite small, a relatively large relative stop criterion is enough to obtain accurate time-dependent solutions after time $t = 0$.
The input file differs in some places from the input file for the direct solver:

```
#  plume2d_iter.prb
#
#  problem file for 2d developing plume problem in the Earth's mantle
#  An iterative linear solver is used
#  integrated (coupled) approach
#  The heat equation and Stokes equation are solved in a decoupled way
#  In each time step corresponding to the solution of the heat equation,
#  the stationary Stokes equations are solved
#
#  See the Examples Manual Section 4.2.1
#
#  To run this file use:
#     sepcomp plume2d_iter.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      Stokes = 1                      # Sequence number of Stokes problem
      heat = 2                        # Sequence number of heat problem
      prlvl       = 0                 # print level in solvers
      inormvel_1  = 2                 # sequence_number of var. normvel_1
      inormvel_2  = 3                 # sequence_number of var. normvel_2
   reals
      eps        = 0                  # penalty parameter for Stokes (not used)
      Ra         = 1d5                # Rayleigh number
      deta       = 1d3                # viscosity contrast for T=0 and T=1
      rho        = 1                  # density
      kappa      = 1                  # heat conduction
```

```
      rho_cp    = 1                 # rho cp
      diffmaxvrel = 1e-2            # rel. conv.crit. Stokes solver
      diffmaxvabs = 1e-2            # abs. conv.crit. Stokes solver
      diffmaxT    = 1e-2            # rel. conv.crit. heateq. solver
   vector_names
      1: velocity                   # Stokes solution
      2: temperature                # solution heat equation
   variables
      1: normvel                    # To store maximum velocity
      2:  normvel_1                 # Stokes-dof1-norm
      3:  normvel_2                 # Stokes-dof2-norm
end
#
#  Define the type of problem to be solved
#
problem $Stokes           # See Users Manual Section 3.2.2
                          # Problem description of the Stokes equation

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=902                # Type number for Navier-Stokes, without swirl
                                # Element type for 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
      degfd2,curves(c1)         # Normal velocity on lower wall prescribed
      curves(c2)                # Fixed right-hand side wall
      degfd2,curves(c3)         # Normal velocity on upper wall prescribed
      degfd1,curves(c4)         # Normal velocity on left-hand side wall
                                # prescribed
                                # All not prescribed boundary conditions
                                # satisfy corresponding stress is zero
   renumber levels (1,2),(3,4,5) # Renumber to enable ILU
                                # Otherwise we might get zeros on the main
                                # diagonal, corresponding to the continuity
                                # equation
problem $heat             # Problem description of the heat equation

   types                        # Define types of elements,
                                # See Users Manual Section 3.2.2
      elgrp1=800                # Type number for heat 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)                # Temperature given on lower wall
      curves(c2)                # Temperature given on right-hand side wall
      curves(c3)                # Temperature given on upper wall
end

#  Define the structure of the large matrices
#  See Users Manual Section 3.2.4
```

```
matrix
   method = 6, problem=$Stokes # Non-symmetrical compact matrix for Stokes
                               # Mark that in this case the symmetry is lost
                               # So an iterative method will be applied
   method = 6, problem=$heat   # Non-symmetrical compact matrix for the heat
                               # equation
                               # 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 a time-dependent problem is coupled with a
#  time-independent problem and also some extra computations are carried out
#
structure                    # See Users Manual Section 3.2.3
   # Create initial conditions:

      create_vector, vector = %velocity, sequence_number 1  # set v = 0
        # Actually setting u = 0, means that the essential boundary conditions
        # are made equal to 0
        # This statement could also be replaced by:
        # prescribe_boundary_conditions, vector = %velocity

   create_vector, vector = %temperature, sequence_number 2  # T given

   # Compute the velocity in the first step by solving the Stokes equations
   # Use a high accuracy to get a good starting value

      solve_linear_system, seq_coef=1, vector = %velocity, seq_solve=3

   # Compute the maximal norm of the velocity over all points in the mesh
   # Store in normvel

      compute_scalar%normvel_1, norm=3, vector1=%velocity, degfd1
      compute_scalar%normvel_2, norm=3, vector1=%velocity, degfd2
      scalar%normvel = max ( S$inormvel_1, S$inormvel_2 )
      compute_scalar%normvel, norm=3, vector1=%velocity

   # Write the initial condition to sepcomp.out

      output, sequence_number=1

   # Time integration:

   start_time_loop

      # solve 1 timestep for T:

         time_integration, sequence_number = 1, vector=%temperature

      # solve stationary v:

         solve_linear_system, seq_coef=1, problem=$Stokes, vector=%velocity //
            seq_solve=1
```

```
      # Compute the max norm of the velocity
      # Store in normvel

         compute_scalar%normvel_1, norm=3, vector1=%velocity, degfd1
         compute_scalar%normvel_2, norm=3, vector1=%velocity, degfd2
         scalar%normvel = max ( S$inormvel_1, S$inormvel_2 )

      # Use the own-written subroutine userout to
      # print the computed max norm and to write to the file vmax.dat

         user_output

      # Write the solution vectors to sepcomp.out at times defined by
      # toutinit, toutstep and toutend

         output, sequence_number=1

   end_time_loop

end

#  Create initial vectors for velocity and Temperature
#  This part describes which values are used
#  See Users Manual Section 3.2.10

create vector, sequence_number 1, problem $Stokes
   value = 0    # The velocity vector is set to 0
end
create vector, sequence_number 2, problem $heat
   curves (c1), func = 1     # The Temperature at the lower wall is a function
                             # of the position: T = 1-2x
                             # This is defined in function func
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 and 3.1

coefficients, sequence_number=1, problem = $Stokes    # problem 1: Stokes
   elgrp1 ( nparm = 20)                         # The coefficients are defined
                                                # by 20 parameters
     icoef2 = 1                                 # constitution eq.
                                                # Newtonian model
     icoef4 = 1                                 # axi-symmetry
     icoef5 = 0                                 # Stokes' flow (no inertia term)
     coef6  = $eps                              # penalty parameter
     coef7  = $rho                              # density of fluid
     coef10 = old solution %temperature, coef=$Ra    # y-dir. body force
     coef12:  sol_func=1                        # eta is a function of the
                                                # temperature and the viscosity
                                                # contrast deta
                                                # Function FUNCC3 is used
end
```

```
# Coefficients for the heat equation

coefficients, sequence_number=2, problem = $heat    # problem 2: heat-eq.
   elgrp1 ( nparm = 20)                            # The coefficients are defined
     icoef3= 3                                     # integration rule
     icoef4= 1                                     # axi-symmetry
     coef6 = $kappa                                # thermal conductivity
     coef9 = $kappa                                # thermal conductivity
     coef12= old solution %velocity degree of freedom 1 # u-velocity
     coef13= old solution %velocity degree of freedom 2 # v-velocity
     coef17= $rho_cp                               # rho cp
end

#  Input for the linear solvers
#  See Users Manual Section 3.2.8


solve, sequence_number = 1   # Solver for Stokes during time stepping
   iteration_method    = gmresr,       //
   preconditioning     = ilu,          //
   print_level         = $prlvl,       //
   start=old_solution,                 //
   abs_accuracy        = $diffmaxvabs, //
   rel_accuracy        = $diffmaxvrel, //
   max_iter=1000
end
solve, sequence_number = 2   # Solver for heat eq. during time stepping
   iteration_method    = gmresr,       //
   preconditioning     = ilu,          //
   print_level         = $prlvl,       //
   start=old_solution,                 //
   accuracy            = $diffmaxT,    //
   max_iter            = 1000
end
solve, sequence_number = 3   # Solver for Stokes during the initial step
   iteration_method    = gmresr,       //
   preconditioning     = ilu,          //
   print_level         = $prlvl,       //
   start=old_solution,                 //
   abs_accuracy        = 1d-5,         //
   rel_accuracy        = 1e-5,         //
   max_iter            = 1000
end

#  Input for the time integration
#  See Users Manual Section 3.2.15

time_integration, sequence_number = 1
   method                  = euler_implicit    # Default time integration scheme
   tinit                   = 0                 # Initial time
   tend                    = 8d-3              # end time
   tstep                   = 1d-4              # time step
   toutinit                = 0                 # Initial time for output
```

```
   toutend               = 8d-3              # end time for output
   toutstep              = 1d-4              # time step for output
   seq_coefficients      = 2                 # Defines which coefficients
                                             # are used for the heat eq.
   boundary_conditions   = initial_field     # The boundary conditions are
                                             # constant and equal to the
                                             # bc's at t=0
   seq_solution_method   = 2                 # Input for linear solver
   print_level           = 4                 # Defines the amount of output
                                       # Mark that in this case a consistent
                                       # mass matrix is used, yielding a
                                       # slightly different solution
end

#  Write both velocity and temperature to output file for postprocessing
#  See Users Manual, Section 3.2.13
#  Not necessary, because this is the default

output, sequence_number=1
  write 2 solutions
end

end_of_sepran_input
```

Using seppost with the following input gives temperature contour plots and velocity vector plots in time:

```
#  plume2d_iter.pst
#
#  Input file for postprocessing for
#  2d developing plume problem in the Earth's mantle
#
#  See the Examples Manual Section 4.2.1
#
#  To run this file use:
#     seppost plume2d_iter.pst > plume2d_iter.out
#
#  Reads the files meshoutput and sepcomp.out
#
#
postprocessing                     # See Users Manual Section 5.2

#  For all time steps do

   time = 0, 8d-3, 1

     # Plot velocity field
      plot vector v%velocity

     # Plot temperature field
      plot coloured contour v%temperature

end
```

Figure 4.2.1.2 gives the plume development at time $t = .0036$.

A 3-D model with iterative solution method

For all but the smallest 3-D meshes, a direct solution method is not feasible anymore, due to enormous computation times, and matrix sizes that are in general too large in comparison with the available computer memory. Therefore, we only give the iterative solution method in this case.

The mesh is defined as a quarter of a cylinder with unit height. Input for sepmesh is given below, and Figure 4.2.1.3 shows the build-up of the mesh.

```
#  plume3d_iter.msh
#
#  mesh file for 3d developing plume problem in the Earth's mantle
#  An iterative linear solver is used
#  See the Examples Manual Section 4.2.1
#
#  To run this file use:
#     sepmesh plume3d_iter.msh
#
#  Creates the file meshoutput
#
#  Define some general constants
#
constants            # See Users Manual Section 1.4
   integers
      nline =  17     # Number of elements in radial dir.
      ncurv =  20     # Number of elements in pipe-surface
      cur_shape = 2   # shape of curve elements (quadratic)
      sur_shape = 6   # shape of surface elements (bi-quadratic quadrilaterals)
      vol_shape =14   # shape of surface elements (bi-quadratic quadrilaterals)
   reals
      f     =  1.05   # rel. element increase per elem. in radial dir.
      length = 1      # Length of region
      width = 1       # Width of region
      height = 1      # Width of region
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 lower face
      p2=($length,0,0)          # Right under point lower face
      p3=(0,$width,0)           # Left upper point lower face
      p4=(0,0,$height)          # Centroid of upper face
      p5=($length,0,$height)    # Right under point upper face
      p6=(0,$width,$height)     # Left upper point upper face
#  curves
#
   curves            # See Users Manual Section 2.3
```

```
      # line in x-direction in lower face
        c1 = line $cur_shape (p1,p2,ratio=2, factor=$f,nelm = $nline)
      # arc in lower face
        c2 = arc $cur_shape (p2,p3,p1,nelm=$ncurv)
      # line in y-direction in lower face
        c3 = line $cur_shape (p3,p1,ratio=4, factor=$f,nelm = $nline)
      # line in x-direction in upper face
        c4 = line $cur_shape (p4,p5,ratio=2, factor=$f,nelm = $nline)
      # arc in upper face
        c5 = arc $cur_shape (p5,p6,p4,nelm=$ncurv)
      # line in y-direction in upper face
        c6 = line $cur_shape (p6,p4,ratio=4, factor=$f,nelm = $nline)
      # line connecting centroids in both faces
        c7 = line $cur_shape (p1,p4,nelm = $nline)
      # line connecting x end points in both faces
        c8 = line $cur_shape (p2,p5,nelm = $nline)
      # line connecting y end points in both faces
        c9 = line $cur_shape (p3,p6,nelm = $nline)
#
#  surfaces
#
   surfaces         # See Users Manual Section 2.4
   s1 = general $sur_shape  (c1, c2, c3 )         # lower surface
   s2 = general $sur_shape  (c4, c5, c6 )         # upper surface
   s3 = pipesurface $sur_shape (c1,c4,c7,c8)      # The pipe surface is
   s4 = pipesurface $sur_shape (c2,c5,c8,c9)      # subdivided into 3 parts
   s5 = pipesurface $sur_shape (c3,c6,c9,c7)
   s6 = ordered surface(s3,s4,s5)                 # Make one surface
#
#  volumes
#
   volumes          # See Users Manual Section 2.5
      v1 = pipe $vol_shape (s1,s2,s6)

   plot, eyepoint = (3, 1.5, 3)    # make a plot of the mesh
                                  # See Users Manual Section 2.2
end
```

The main program is also a little bit different from the 2d version:

```
      program plume3d_iter

c     --- Main program for 3d  developing plume problem in the Earth's mantle
c         An iterative linear solver is used
c         integrated (coupled) approach
c         The heat equation and Stokes equation are solved in a decoupled way
c         In each time step corresponding to the solution of the heat equation,
c         the stationary Stokes equations are solved
c         The main program is necessary because function subroutines are used
c         Also it is necessary to use a larger buffer

      integer ibuffr, pbuffr
      parameter (pbuffr=100000000)
```

```
      common ibuffr(pbuffr)

      call sepcom(pbuffr)

      end

c     --- Function func is used to define the initial temperature along the
c         lower wall

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

      if ( ichoice.eq.1 ) then

c     --- initial temperature distribution:

         rad = sqrt(x**2 + y**2)
         func = max(0d0, 1d0 - 2d0*rad)

      else

c     --- Other values of ichoice not permitted

         print *,'This value of ichoice (',ichoice,') has not been ',
     +           'implemented in function func'

      end if

      end


c     --- Function funcc3 is used to define the viscosity as function
c         of the temperature and the viscosity contrast for T=0 and T=1

      function funcc3 ( ifunc, x, y, z, numold, maxunk, uold )
      implicit none
      integer ifunc, numold, maxunk
      double precision funcc3, x, y, z, uold(numold,maxunk)

      integer iold, icount
      double precision deta, getconst

c     --- Make the following variables valid in all calls
c         Their values are kept

      save iold, icount, deta

c     --- Initialize icount to 0

      data icount /0/

      if ( ifunc.eq.1 ) then
```

```
c     --- ifunc = 1, compute viscosity

         if ( icount.eq.0 ) then

c        --- First step, get values of iold and deta
c            Since these parameters are saved, they have to computed only once
c            Get sequence number of temperature in solution vector
c            and store in iold
c            Get deta from the input file

            call prgetname ( 'temperature', iold )
            deta=getconst('deta')

c           --- change icount in order to prevent recomputing these constants

            icount = 1

         end if

c        --- Compute viscosity

         funcc3 = exp(-log(deta) * uold(iold,1) )

      else

c     --- Other values of ichoice not permitted

         print *,'This value of ifunc (',ifunc,') has not been ',
     +           'implemented in function funcc3'

      end if


      end

c     --- Subroutine userout is used to store the computed maximum
c         velocity into the file vmax.dat for postprocessing purposes

      subroutine userout ( kmesh, kprob, isol, isequence, numvec )
      implicit none
      include 'SPcommon/comcons1'
      include 'SPcommon/cuscons'
      include 'SPcommon/ctimen'

      integer kmesh(*), kprob(*), numvec, isol(5,*), isequence(*)

      double precision getvar

      integer icount, iref
      double precision vmax

c     --- Save icount so that the value is kept for a subsequent call
```

```
      save icount

c     --- Initialize icount to 0

      data icount /0/

      iref = 31
      if ( icount.eq.0 ) then

c     --- Open the file vmax.dat in the first step (icount=0)
c         Since icount is saved, this is an allowed construction

         open ( unit=iref, file='vmax.dat' )
         icount = 1

      end if
      vmax = getvar('normvel')
      write(iref,*) t, vmax

      write(*,100)' time=', t,': Vmax=', vmax
100   format(a,f10.5,a,f10.5)

      end
```

Input for the computational program is given below:

```
#  plume3d_iter.prb
#
#  problem file for 3d developing plume problem in the Earth's mantle
#  An iterative linear solver is used
#  integrated (coupled) approach
#  The heat equation and Stokes equation are solved in a decoupled way
#  In each time step corresponding to the solution of the heat equation,
#  the stationary Stokes equations are solved
#
#  See the Examples Manual Section 4.2.1
#
#  To run this file use:
#     sepcomp plume3d_iter.prb
#
#  Reads the file meshoutput
#  Creates the file sepcomp.out
#
#
#
#  Define some general constants
#
constants           # See Users Manual Section 1.4
   integers
      Stokes = 1                      # Sequence number of Stokes problem
      heat = 2                        # Sequence number of heat problem
      prlvl       = 3                 # print level in solvers
      inormvel_1  = 3                 # sequence_number of var. normvel_1
      inormvel_2  = 4                 # sequence_number of var. normvel_2
```

```
      inormvel_3  = 5     # scalar nr. normvel
   reals
      eps       = 0                 # penalty parameter for Stokes (not used)
      Ra        = 1d5               # Rayleigh number
      deta      = 1d3               # viscosity contrast for T=0 and T=1
      rho       = 1                 # density
      kappa     = 1                 # heat conduction
      rho_cp    = 1                 # rho cp
      diffmaxv  = 1e-4              # conv.crit. Stokes solver
      diffmaxT  = 1e-4              # conv.crit. heateq. solver
   vector_names
      1: velocity                   # Stokes solution
      2: temperature                # solution heat equation
   variables
      1:  diff        = 2     # error in Picard iter.
      2:  normvel             # Stokes-norm in Picard
      3:  normvel_1           # Stokes-dof1-norm in Picard
      4:  normvel_2           # Stokes-dof2-norm in Picard
      5:  normvel_3           # Stokes-dof2-norm in Picard
end
#
#  Define the type of problem to be solved
#
problem $Stokes          # See Users Manual Section 3.2.2
                         # Problem description of the Stokes equation

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=902               # Type number for Navier-Stokes, without swirl
                               # Element type for 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
      degfd3,surfaces(s1)      # bottom (normal component)
      degfd3,surfaces(s2)      # top (normal component)
      degfd2,surfaces(s3)      # front surface (normal component)
      surfaces(s4)             # pipe surface (completely prescribed)
      degfd1,surfaces(s5)      # left-hand side face (normal component)
                               # All not prescribed boundary conditions
                               # satisfy corresponding stress is zero
   renumber levels (1,2,3),(4,5,6,7) # Renumber to enable ILU
                               # Otherwise we might get zeros on the main
                               # diagonal, corresponding to the continuity
                               # equation
problem $heat            # Problem description of the heat equation

   types                       # Define types of elements,
                               # See Users Manual Section 3.2.2
      elgrp1=800               # Type number for heat 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
```

```
      surfaces(s1)             # Temperature given on bottom
      surfaces(s2)             # Temperature given on top
      surfaces(s4)             # Temperature given on pipe surface
end

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

matrix
   method = 6, problem=$Stokes # Non-symmetrical compact matrix for Stokes
                               # Mark that in this case the symmetry is lost
                               # So an iterative method will be applied
   method = 6, problem=$heat   # Non-symmetrical compact matrix for the heat
                               # equation
                               # 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 a time-dependent problem is coupled with a
#  time-independent problem and also some extra computations are carried out
#
structure                   # See Users Manual Section 3.2.3
   # Create initial conditions:

      create_vector, vector = %velocity, sequence_number 1  # set v = 0
        # Actually setting u = 0, means that the essential boundary conditions
        # are made equal to 0
        # This statement could also be replaced by:
        # prescribe_boundary_conditions, vector = %velocity

      create_vector, vector = %temperature, sequence_number 2  # T given

   # Compute the velocity in the first step by solving the Stokes equations
   # Use a high accuracy to get a good starting value

      solve_linear_system, seq_coef=1, vector = %velocity, seq_solve=3

   # Compute the maximal norm of the velocity over all points in the mesh
   # Store in normvel

      compute_scalar%normvel_1, norm=3, vector1=%velocity, degfd1
      compute_scalar%normvel_2, norm=3, vector1=%velocity, degfd2
      compute_scalar%normvel_3, norm=3, vector1=%velocity, degfd3
      scalar%normvel = max ( S$inormvel_1, S$inormvel_2, S$inormvel_3 )

   # Use the own-written subroutine userout to
   # print the computed max norm and to write to the file vmax.dat

      user_output

   # Write the initial condition to sepcomp.out

      output, sequence_number=1
```

```
   # Time integration:

   start_time_loop

      # solve 1 timestep for T:

         time_integration, sequence_number = 1, vector=%temperature

      # solve stationary v:

         solve_linear_system, seq_coef=1, problem=$Stokes, vector=%velocity //
            seq_solve=1

      # Compute the max norm of the velocity
      # Store in normvel

         compute_scalar%normvel_1, norm=3, vector1=%velocity, degfd1
         compute_scalar%normvel_2, norm=3, vector1=%velocity, degfd2
         compute_scalar%normvel_3, norm=3, vector1=%velocity, degfd3
         scalar%normvel=max(S$inormvel_1,S$inormvel_2,S$inormvel_3)

      # Use the own-written subroutine userout to
      # print the computed max norm and to write to the file vmax.dat

         user_output

      # Write the solution vectors to sepcomp.out at times defined by
      # toutinit, toutstep and toutend

         output, sequence_number=1

   end_time_loop

end

#  Create initial vectors for velocity and Temperature
#  This part describes which values are used
#  See Users Manual Section 3.2.10

create vector, sequence_number 1, problem $Stokes
   value = 0    # The velocity vector is set to 0
end
create vector, sequence_number 2, problem $heat
   surfaces (s1), func = 1   # The Temperature at the lower wall is a function
                             # of the position: T = 1-2x
                             # This is defined in function func
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 and 3.1

coefficients, sequence_number=1, problem = $Stokes    # problem 1: Stokes
```

```
   elgrp1 ( nparm = 20)                           # The coefficients are defined
                                                  # by 20 parameters
     icoef2 = 1                                   # constitution eq.
                                                  # Newtonian model
     icoef5 = 0                                   # Stokes' flow (no inertia term)
     coef6  = $eps                                # penalty parameter
     coef7  = $rho                                # density of fluid
     coef11 = old solution %temperature, coef=$Ra    # z-dir. body force
     coef12:  sol_func=1                          # eta is a function of the
                                                  # temperature and the viscosity
                                                  # contrast deta
                                                  # Function FUNCC3 is used
end

# Coefficients for the heat equation

coefficients, sequence_number=2, problem = $heat    # problem 2: heat-eq.
   elgrp1 ( nparm = 20)                           # The coefficients are defined
     icoef3= 3                                    # integration rule
     coef6 = $kappa                               # thermal conductivity
     coef9 = $kappa                               # thermal conductivity
     coef11= $kappa                               # thermal conductivity
     coef12= old solution %velocity degree of freedom 1 # u-velocity
     coef13= old solution %velocity degree of freedom 2 # v-velocity
     coef14= old solution %velocity degree of freedom 3 # w-velocity
     coef17= $rho_cp                              # rho cp
end

#  Input for the linear solvers
#  See Users Manual Section 3.2.8


solve, sequence_number = 1   # Solver for Stokes during time stepping
   iteration_method    = gmresr,       //
   preconditioning     = ilu,          //
   print_level         = $prlvl,       //
   start               = old_solution, //
   accuracy            = $diffmaxv,    //
   max_iter            = 1000
end
solve, sequence_number = 2   # Solver for heat eq. during time stepping
   iteration_method    = gmresr,       //
   preconditioning     = ilu,          //
   print_level         = $prlvl,       //
   start               = old_solution, //
   accuracy            = $diffmaxT,    //
   max_iter            = 1000
end
solve, sequence_number = 3   # Solver for Stokes during the initial step
   iteration_method    = gmresr,       //
   preconditioning     = ilu,          //
   print_level         = $prlvl,       //
   start               = old_solution, //
   abs_accuracy        = 1d-5,         //
```

```
   rel_accuracy        = 1d-5,         //
   max_iter            = 1000
end

#  Input for the time integration
#  See Users Manual Section 3.2.15

time_integration, sequence_number = 1
   method                  = euler_implicit    # Default time integration scheme
   tinit                   = 0                 # Initial time
   tend                    = 8d-3              # end time
   tstep                   = 1d-4              # time step
   toutinit                = 0                 # Initial time for output
   toutend                 = 8d-3              # end time for output
   toutstep                = 1d-4              # time step for output
   seq_coefficients        = 2                 # Defines which coefficients
                                               # are used for the heat eq.
   boundary_conditions     = initial_field     # The boundary conditions are
                                               # constant and equal to the
                                               # bc's at t=0
   seq_solution_method     = 2                 # Input for linear solver
   print_level             = 4                 # Defines the amount of output
end

#  Write both velocity and temperature to output file for postprocessing
#  See Users Manual, Section 3.2.13
#  Not necessary, because this is the default

output, sequence_number=1
  write 2 solutions
end

end_of_sepran_input
```

To visualize the 3-D results, we plot 2-D cross sections through the 3-D domain. In this case, the cross sections are chosen parallel to the plain $x = 0$. For the cross section at $x = 0$, temperature results are similar to the ones showed in Figure 4.2.1.2.

```
#  plume3d_iter.pst
#
#  Input file for postprocessing for
#  3d developing plume problem in the Earth's mantle
#
#  See the Examples Manual Section 4.2.1
#
#  To run this file use:
#     seppost plume3d_iter.pst > plume3d_iter.out
#
#  Reads the files meshoutput and sepcomp.out
#
#
postprocessing                    # See Users Manual Section 5.2

#  For all time steps do
```

```
   time = 0, 8d-3, 1

     # compute intersection with plane x=0
        compute v%vel_intersect = intersection v%velocity, plane(x=0.0) //
           numbunknowns=3, transformation=plane_oriented
        compute v%temp_intersect = intersection v%temperature, plane(x=0.0)

     # Plot velocity field
         plot vector v%vel_intersect, norotate

     # Plot temperature field
         plot coloured contour v%temp_intersect, norotate
end
```

Figure 4.2.1.4 shows the maximum velocity in time for both the 2-D and the 3-D calculation.

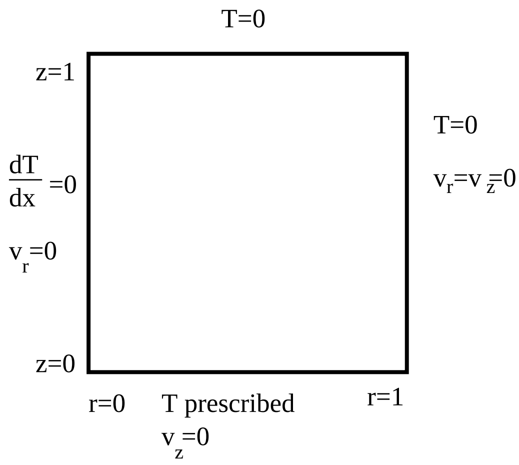

Figure 4.2.1.1: General flow and thermal boundary conditions for mantle plume model

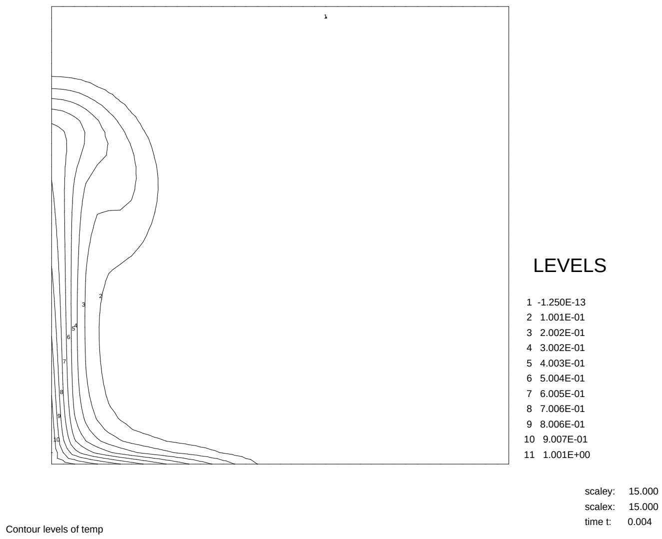

Figure 4.2.1.2: Temperature contour plot of the developing mantle plume at time $t = 0.0036$

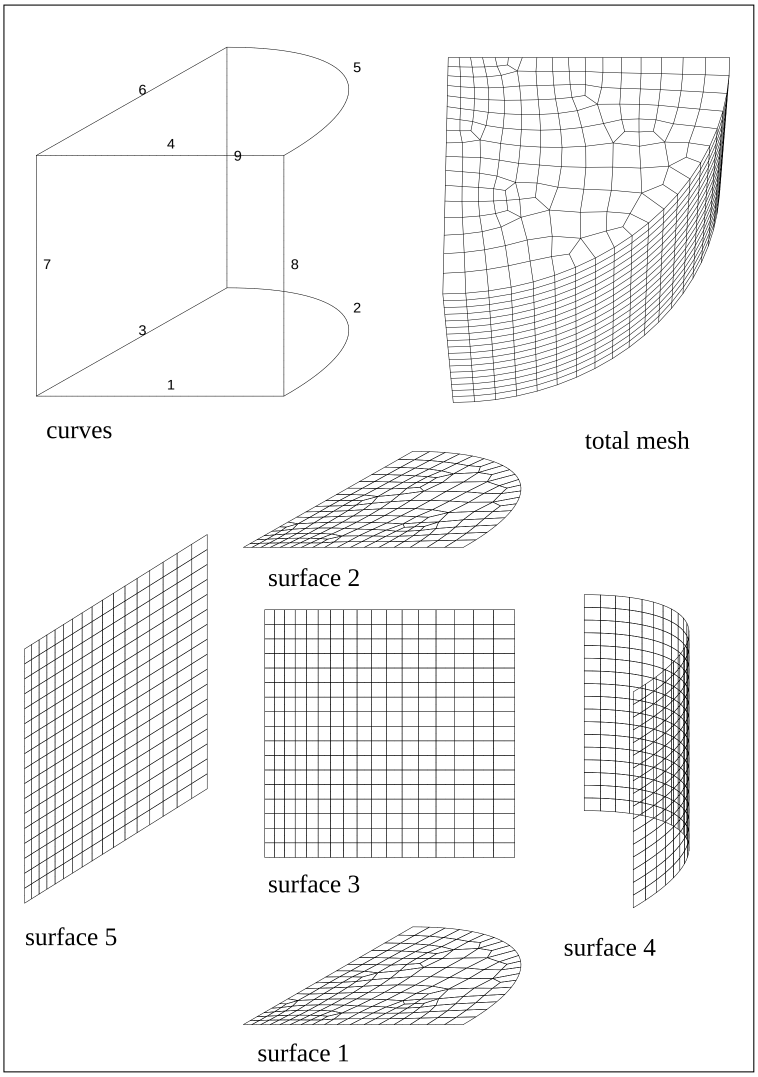

Figure 4.2.1.3: 3-D mesh for rising mantle plume

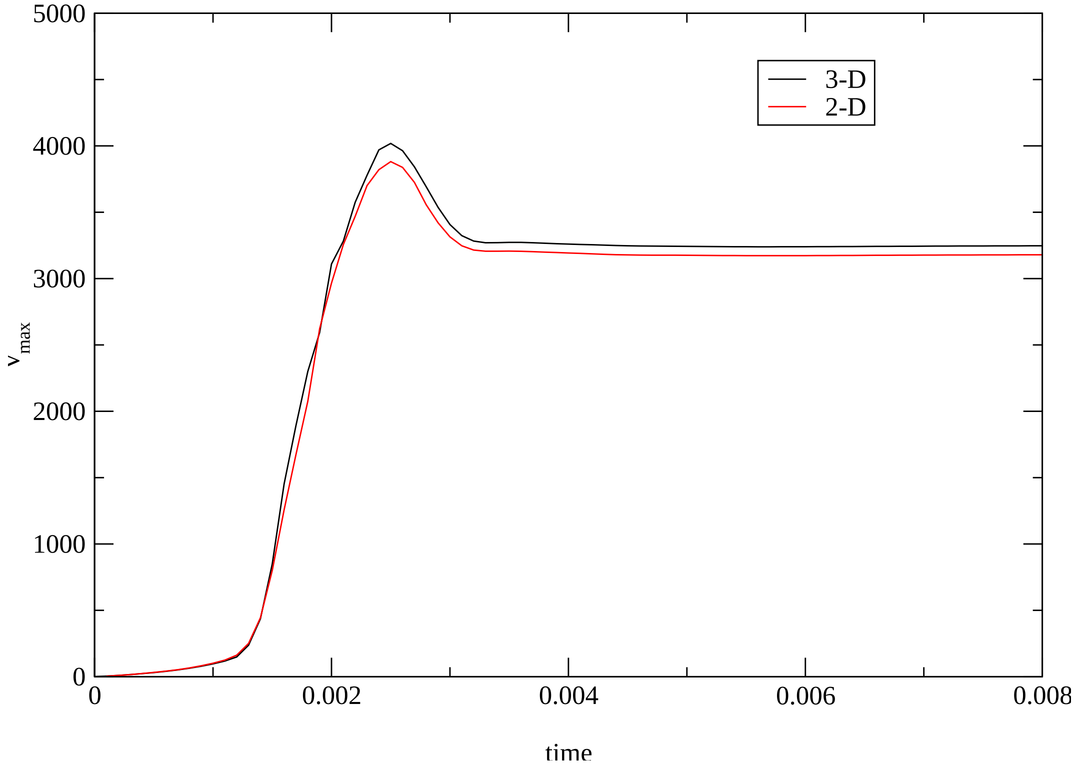

Figure 4.2.1.4: Maximum velocity v_{max} for both the 2-D and the 3-D calculation

5 References

van Brummelen, H. (2001) To appear

Cahouet, J. (1984) Etude Numérique et Experimentale du Problème Bidimensionnel de la Résistance de Vagues Non-Linéaire", ENSTA, Paris, (In French)

Darwish, P.L.J., Zitha, J.R.C., van der Maarel, L. Pel (2001) Polymer gel barriers for waste disposal facilities, In proceedings of the Fourteenth Southeast Asian Geotechnical Conference, Hong Kong **December 10-14**.

Logan, J.D. (1994) An introduction to nonlinear partial differential equations, (Wiley, New York)

Segal, Guus and Kees Vuik (1995) A simple iterative linear solver for the 3D incompressible Navier-Stokes equations discretized by the finite element method.
Faculty of Technical Mathematics and Informatics
Delft University of Technology,1995
TUD Report 95-64.

6 Index