Programmers Guide of the ISNaS incompressible flow solver

Guus Segal

version 1.0

August 16, 2000

Contents

1 Introduction **5**

2 Input files **7**
2.1 Introduction . . . 7
2.2 Description of some of the input files . . . 7
2.2.1 File finvol.new . . . 7
2.2.2 File isnasinp.cmp . . . 12
2.2.3 File isnas.hosts . . . 27
2.2.4 File isnas.env . . . 27
2.2.5 File isnas.dbg . . . 33

3 Output files **35**
3.1 Introduction . . . 35
3.2 Description of some of the output files . . . 35
3.2.1 File isnascomp.out . . . 35
3.2.2 File isnasback . . . 37

4 The structure of the main program **39**
4.1 Introduction . . . 39
4.2 Structure of the host program . . . 39
4.2.1 Subroutine isnmainh . . . 39
4.3 Structure of the node program . . . 39
4.4 Detailed description of some general starting and closing subroutines . . . 39
4.4.1 Subroutine ishstart . . . 39
4.4.2 Subroutine isnstart . . . 40
4.4.3 Subroutine isfinish . . . 42
4.4.4 Subroutine isinitcb . . . 42
4.4.5 Subroutine isinitmd . . . 42
4.4.6 Subroutine isinifil . . . 43
4.4.7 Subroutine isacth . . . 43
4.5 Organization of the multi block process in a parallel environment . . . 43

5 Memory management **47**
5.1 Introduction . . . 47
5.2 Tasks of the memory management system . . . 47
5.3 An overview of the memory management tools . . . 48
5.4 How to use the memory management system . . . 48
5.5 The structure of array ibuffr . . . 50
5.6 Detailed description of the memory management tools . . . 51
5.6.1 Subroutine isstarmm . . . 51
5.6.2 Subroutine isactive . . . 51
5.6.3 Subroutine isicreat . . . 52

5.6.4 Subroutine isdcreat . . . 52
5.6.5 Subroutine isiexten . . . 53
5.6.6 Subroutine isdexten . . . 53
5.6.7 Subroutine istmspac . . . 54
5.6.8 Subroutine isdelarr . . . 54
5.6.9 Subroutine isfreesp . . . 55
5.6.10 Subroutine isendspc . . . 55
5.6.11 Subroutine islreset . . . 56
5.6.12 Subroutine isdcrpri . . . 56
5.6.13 Subroutine issetpri . . . 56
5.6.14 Subroutine isingetp . . . 57
5.6.15 Subroutine isdpgetp . . . 57
5.6.16 Subroutine isinleng . . . 58
5.6.17 Subroutine isdpleng . . . 58

6 Parallel aspects **59**
6.1 introduction . . . 59
6.2 Description of the communication subroutines . . . 59
6.2.1 Subroutine istparhs . . . 59
6.2.2 Subroutine istparnd . . . 60
6.2.3 Subroutine issend . . . 60
6.2.4 Subroutine isreceiv . . . 62
6.2.5 Subroutine isexitpm . . . 63
6.3 Common block cisparal . . . 63
6.4 Organization of the send buffer in case of parallel processing . . . 64
6.4.1 Common block ciscom . . . 65

7 Error messages **67**
7.1 Introduction . . . 67
7.2 Definition of the error message file. . . . 68
7.3 Creation and updating of the error message file. . . . 68
7.4 Agreements with respect to the filling of the error message file. . . . 70
7.5 Available subroutines with respect to error messages and warnings . . . 70
7.5.1 Subroutine iserrsub . . . 71
7.5.2 Subroutine iswarsub . . . 72
7.5.3 The common blocks cismessc and cismessg . . . 72
7.5.4 Subroutine iserrint . . . 73
7.5.5 Subroutine iserreal . . . 73
7.5.6 Subroutine iserrchr . . . 74
7.5.7 Subroutine iseropen . . . 74
7.5.8 Subroutine iserclos . . . 75
7.5.9 Subroutine iserclmn . . . 75
7.6 Special arrangements with respect to parallel processing . . . 76

8 Timing subroutines **77**
8.1 Introduction . . . 77
8.2 Description of the timing subroutines . . . 77
8.2.1 Subroutine isiniclk . . . 77
8.2.2 Subroutine issetclk . . . 78
8.2.3 Subroutine isaskclk . . . 78
8.3 Common block cistime . . . 79

9 Print subroutines **81**
9.1 Introduction . . . 81
9.2 General print subroutines . . . 81
9.2.1 Subroutine isprint . . . 81
9.2.2 Subroutine isgenpr . . . 82
9.2.3 Subroutine isuserpr . . . 83
9.3 Special print subroutines . . . 84
9.3.1 Subroutine isprar01 . . . 84
9.3.2 Subroutine isprar02 . . . 85
9.3.3 Subroutine isprar03 . . . 85
9.3.4 Subroutine isprar04 . . . 86
9.3.5 Subroutine isprimt1 . . . 87
9.3.6 Subroutine isprimt2 . . . 87
9.3.7 Subroutine isprirh1 . . . 88
9.3.8 Subroutine isprirh2 . . . 89

10 Linear solvers **91**

11 Building of matrices and right-hand sides **93**
11.1 Introduction . . . 93
11.2 Storage and Administration . . . 93
11.2.1 Evaluation of the inner cells . . . 95
11.2.2 Evaluation of the boundary conditions . . . 98
11.2.3 Some notation and conventions for the place in the grid . . . 99
11.2.4 Interpolation and finite difference formulae . . . 101
11.2.5 The matrix information arrays . . . 104
11.3 Reference structure . . . 106
11.3.1 Reference structure of the general matrix assembly subroutine . . . 106
11.3.2 Reference structure of the momentum matrix assembly subroutine . . . 106

12 Time integration **109**

13 The multiblock process **111**

14 General tools **113**
14.1 Introduction . . . 113
14.2 Copy subroutines . . . 113

15 Debugging possibilities **115**
15.1 Introduction . . . 115

16 Data structures **117**
16.1 introduction . . . 117
16.2 Sequence of the unknowns . . . 119
16.3 The co-ordinates arrays . . . 119
16.4 The solution arrays . . . 120
16.5 Geometrical quantities . . . 121
16.6 The matrices . . . 122
16.7 The right-hand side vectors . . . 130
16.8 The matrix information arrays INFMM, INFMG, INFMT . . . 130
16.9 The input arrays IINPUT and RINPUT . . . 131
16.10 The coefficients arrays . . . 135
16.11 Arrays with respect to the boundary conditions . . . 137
16.11.1 Array INFBND . . . 137
16.11.2 Array BNDCON . . . 139

16.11.3 Array IINBC . . . 139
16.11.4 Array RINBC . . . 142
16.12 Arrays with respect to the initial condition . . . 142
16.12.1 Array IINCND . . . 142
16.12.2 Array RINCND . . . 142
16.13 Arrays with respect to the time integration . . . 143
16.13.1 Array IINTIM . . . 143
16.13.2 Array RINTIM . . . 144
16.14 Arrays with respect to the turbulence modeling . . . 144
16.14.1 Array IINTUR . . . 145
16.14.2 Array RINTUR . . . 145
16.14.3 Array EDDY . . . 146
16.15 Arrays with respect to the linear solver . . . 147
16.15.1 Array IINSOL . . . 147
16.15.2 Array RINSOL . . . 149
16.16 Arrays with respect to the discretization . . . 149
16.16.1 Array IINDSC . . . 149
16.16.2 Array RINDSC . . . 152
16.17 Output arrays . . . 152
16.18 Arrays with respect to the block administration and block algorithm . . . 153
16.18.1 Array IADMIN . . . 153
16.18.2 Array ITOPOL . . . 153
16.18.3 Array SMSG . . . 155
16.18.4 Array IINBLK . . . 156
16.18.5 Array RINBLK . . . 158
16.18.6 Array IACTION . . . 158
16.18.7 Constant ITRANS . . . 159
16.19 Arrays with respect to compressible flow . . . 159
16.19.1 Array IINCOM . . . 159
16.19.2 Array RINCOM . . . 160
16.19.3 Array EDDY . . . 161
16.20 Arrays with respect to the sequence of the arrays . . . 161
16.21 Arrays with respect to the cavity model . . . 161
16.21.1 Array IINCAV . . . 161
16.21.2 Array RINCAV . . . 162
16.22 Arrays with respect to the free surface . . . 162
16.22.1 Array IINFREESURF . . . 162
16.22.2 Array RINFREESURF . . . 162
16.23 Arrays with respect to the structure of the program . . . 163
16.23.1 Array ICLUSTER . . . 163
16.23.2 Array IGLOBSTRUC . . . 163
16.23.3 Array RGLOBSTRUC . . . 164
16.23.4 Array ILOCSTRUC . . . 164
16.23.5 Array RLOCSTRUC . . . 165
16.24 Arrays with respect to flow around profiles . . . 165
16.24.1 Array IINPROFILE . . . 165
16.24.2 Array RINPROFILE . . . 165
16.25 Arrays with respect to frequent output . . . 166
16.25.1 Array IFREQOUT . . . 166
16.26 Common block CISNAS . . . 166
16.27 Common block CUSER . . . 168

Chapter 1

Introduction

This document describes the tools, subroutines, files and data structures that are used by the ISNaS incompressible flow solver. It is only meant for developers of the ISNaS code.

Chapter 2

Input files

2.1 Introduction

In this chapter we describe the various input files that are required by the ISNaS incompressible program.

It concerns the following files:

finvol.new 2.2.1	Contains the description of the grid.
isnasinp.cmp 2.2.2	Contains the intermediate file between pre-processor and computational program.
error.dat (Section 7.2)	ASCII file containing the error messages
errormsg (Section 7.2)	Unformatted direct access file containing the error messages.
isnas.host 2.2.3	Contains the computer names of the computers that are available for parallel processing.
isnas.env 2.2.4	Contains the machine-dependent quantities.
isnas.dbg 2.2.5	Contains information about the debug possibilities.

The names of the files isnas.hosts isnas.env, isnas.dbg and error.dat are fixed and can not be changed by the programmer, without changing the source code of ISNaS. The names of the other files are given in the file isnas.env. These names may be changed by using a local file isnas.env or by changing the global file isnas.env.

The files error.dat and errormsg are described in Section 7.2. All other files are described in Section 2.2.

2.2 Description of some of the input files

In this section we give a complete description of the files: finvol.new, isnasinp.cmp, isnas.hosts, isnas.env and isnas.dbg.

2.2.1 File finvol.new

This section describes the structure of the file finvol.new. This file contains the complete description of the grid including the topology as described in the array itopol (see 16.18.2).

`finvol.new` may be both formatted and unformatted. Whether the file is considered formatted or unformatted depends on the contents of the file `isnas.env`. See 2.2.4

In the formatted case the syntax for the finvol.new file is very simple.

- All records with a $*$ in the first column are considered as comment records.
- All input in a record after a hash-symbol (#) is considered as comment.
- The input consists of records of at most 80 columns.
- The file finvol.new is considered as a standard input file.
- Numbers may be given in any standard FORTRAN format. A real may be given as an integer. Spaces in a number are considered as separators.
- Any character or set of characters which obviously does not belong to the number may be used as separator. As a consequence the text ndim = 2, is read as 2 and the text f2 = 3 as two numbers 2 and 3.

The file finvol.new is subdivided into five parts as follows:

Part number:	Contents
1	The first record must contain the keyword sepran or liss depending on the type of grid generator used
2	general information about the decomposition of the domain and about the sizes of the blocks
3	information needed for the allocation of the ITOPOL array
4	the ITOPOL array
5	a dump of all the grid points

A part may consist of one or more records. The position of the parts as well as the parameters in the parts is fixed. In the next sections the contents of the various parts is described.

Part 1: general information

This part contains general information about the decomposition of the domain and about the sizes of the blocks.

It has the following contents:

```
ndim             # number of dimensions of the problem
nblocks          # number of blocks the domain is decomposed into

ncells(1,1)       ... ncells(1,ndim)        # size of block 1
ncells(2,1)       ... ncells(2,ndim)        # size of block 2
...               ...
ncells(nblocks,1) ... ncells(nblocks,ndim)  # size of last block
```

Part 2: information needed for the allocation of the ITOPOL array

This part contains some parameters which define the sizes from which the size of the ITOPOL array may be computed.

It has the following contents:

```
maxnsubfaces     # maximum number of subfaces encountered on a face
```

```
lensubface       # number of integers that are given in
                 # the description of a subface
                 # Note that the actual itopol array as used in ISNaS
                 # will contain more than lensubface integers for each
                 # subface. (currently lensubface = 7 in the input file)
```

Part 3: the ITOPOL array

This part contains the complete contents of the ITOPOL array.

The contents of this part are given by means of a kind of structure diagram (pseudo code). The FOR loops in this diagram indicate the loops for the input, they must not be given in the input itself.

```
FOR iblock = 1 TO nblocks

   # Note: in the input file, the subfaces are number 1..2*ndim
   #       In the 2-D case, iface has the following meaning
   #          1 --> LEFT
   #          2 --> RIGHT
   #          3 --> DOWN
   #          4 --> UP
   FOR iface = 1 TO 2*ndim

      # PART 3.1: the nsubfaces array

      nsubfaces    # number of subfaces for this face

      # PART 3.2: the subfaces array

      FOR isubface = 1, nsubfaces

         i1           # [i1,i2] x [j1, j2] is the cell index domain of a
         i2           # subface. Note:  j1 = j2 = 0 for 2-D problems
         j1
         j2
         blockno      # Number of neighbouring block (0 if external)
         faceno       # Number of neighbouring face  (0 if external)
         subfaceno    # Number of neighbouring subface (0 if external)
```

With respect to the tangential directions in 3D the following conventions will be used throughout the program:

Face	normal_dir	normal_sign	tang_dir	tang_sign
1	1	-1	(2,3)	(-1,1)
2	1	1	(2,3)	(1,1)
3	2	-1	(3,1)	(-1,1)
4	2	1	(3,1)	(1,1)
5	3	-1	(1,2)	(-1,1)
6	3	1	(1,2)	(1,1)

Part 4: a dump of all the grid points

In this part the co-ordinates of all grid points must be provided. Again the input will be given in the form of a structure diagram. In case of 2D the input is as follows:

```
FOR iblock = 1 TO nblocks

   FOR idim = 1, 2

      FOR iy = 1 TO ny+1

         FOR ix = 1 TO nx+1

            coor(ix,iy,idim)
```

and in case of 3D:

```
FOR iblock = 1 TO nblocks

   FOR idim = 1, ndim

      FOR iz = 1 TO nz+1

         FOR iy = 1 TO ny+1

            FOR ix = 1 TO nx+1

               coor(ix,iy,iz,idim)
```

A sample finvol.new file

In this section we present without any further comment an example of a formatted finvol.new file for a very simple problem.

```
#
#   example of a file finvol.new, for two blocks
#
#   First keyword indicating the package

sepran

#   Part 1: general information

   2                  # ndim
   2                  # nblocks
   5 5                # size of block 1
```

```
   5 5                # size of block 2

#   Part 2: information about sizes

   1                  # maxnsubfaces
   7                  # lensubface

#   Part 3: itopol

#   Block 1
#       Face 1
      1               # nsubfaces
      1 5 0 0 0 0 0   # i1 i2 j1 j2 blockno faceno subfaceno
#       Face 2
      1               # nsubfaces
      1 5 0 0 2 1 1   # i1 i2 j1 j2 blockno faceno subfaceno
#       Face 3
      1               # nsubfaces
      1 5 0 0 0 0 0   # i1 i2 j1 j2 blockno faceno subfaceno
#       Face 4
      1               # nsubfaces
      1 5 0 0 0 0 0   # i1 i2 j1 j2 blockno faceno subfaceno

#   Block 2
#       Face 1
      1               # nsubfaces
      1 5 0 0 1 2 1   # i1 i2 j1 j2 blockno faceno subfaceno
#       Face 2
      1               # nsubfaces
      1 5 0 0 0 0 0   # i1 i2 j1 j2 blockno faceno subfaceno
#       Face 3
      1               # nsubfaces
      1 5 0 0 0 0 0   # i1 i2 j1 j2 blockno faceno subfaceno
#       Face 4
      1               # nsubfaces
      1 5 0 0 0 0 0   # i1 i2 j1 j2 blockno faceno subfaceno

#   Part 4: co-ordinates

#   Block 1

#   x-coordinates

0.000000E+00 2.000000E-01 4.000000E-01 6.000000E-01 8.000000E-01 1.000000E+00
0.000000E+00 2.000000E-01 4.000000E-01 6.000000E-01 8.000000E-01 1.000000E+00
0.000000E+00 2.000000E-01 4.000000E-01 6.000000E-01 8.000000E-01 1.000000E+00
0.000000E+00 2.000000E-01 4.000000E-01 6.000000E-01 8.000000E-01 1.000000E+00
0.000000E+00 2.000000E-01 4.000000E-01 6.000000E-01 8.000000E-01 1.000000E+00
0.000000E+00 2.000000E-01 4.000000E-01 6.000000E-01 8.000000E-01 1.000000E+00

#   y-coordinates

0.000000E+00 0.000000E+00 0.000000E+00 0.000000E+00 0.000000E+00 0.000000E+00
4.000000E-01 4.000000E-01 4.000000E-01 4.000000E-01 4.000000E-01 4.000000E-01
```

```
8.000000E-01 8.000000E-01 8.000000E-01 8.000000E-01 8.000000E-01 8.000000E-01
1.200000E+00 1.200000E+00 1.200000E+00 1.200000E+00 1.200000E+00 1.200000E+00
1.600000E+00 1.600000E+00 1.600000E+00 1.600000E+00 1.600000E+00 1.600000E+00
2.000000E+00 2.000000E+00 2.000000E+00 2.000000E+00 2.000000E+00 2.000000E+00

#   Block 2

#   x-coordinates

1.000000E+00 1.200000E+00 1.400000E+00 1.600000E+00 1.800000E+00 2.000000E+00
1.000000E+00 1.200000E+00 1.400000E+00 1.600000E+00 1.800000E+00 2.000000E+00
1.000000E+00 1.200000E+00 1.400000E+00 1.600000E+00 1.800000E+00 2.000000E+00
1.000000E+00 1.200000E+00 1.400000E+00 1.600000E+00 1.800000E+00 2.000000E+00
1.000000E+00 1.200000E+00 1.400000E+00 1.600000E+00 1.800000E+00 2.000000E+00
1.000000E+00 1.200000E+00 1.400000E+00 1.600000E+00 1.800000E+00 2.000000E+00

#   y-coordinates

0.000000E+00 0.000000E+00 0.000000E+00 0.000000E+00 0.000000E+00 0.000000E+00
4.000000E-01 4.000000E-01 4.000000E-01 4.000000E-01 4.000000E-01 4.000000E-01
8.000000E-01 8.000000E-01 8.000000E-01 8.000000E-01 8.000000E-01 8.000000E-01
1.200000E+00 1.200000E+00 1.200000E+00 1.200000E+00 1.200000E+00 1.200000E+00
1.600000E+00 1.600000E+00 1.600000E+00 1.600000E+00 1.600000E+00 1.600000E+00
2.000000E+00 2.000000E+00 2.000000E+00 2.000000E+00 2.000000E+00 2.000000E+00
```

2.2.2 File isnasinp.cmp

In this section the intermediate file isnasinp.cmp is described. The task of this file is to store all user input. The file is used by the ISNaS processor and created by a ISNaS pre-processor. It will also be possible that the interactive pre-processor reads this file as input for an update of the user input.
At this moment the intermediate file is a formatted file.

The intermediate file contains data and may be made semi-readable by introducing comments. With semi-readable it is meant that the file can be read, but that the interpretation may be difficult.
In order to make the file really readable a separate program will be made that translates the input file into a standard batch user input file, which is of course readable for any user.

With respect to the syntax of this file we follow exactly the same rules as for the intermediate file finvol.new. Hence:

- All records with a $*$ in the first column are considered as comment records.
- All input in a record after a hash-symbol (#) is considered as comment.
- The input consists of records of at most 80 columns.
- The intermediate file is considered as a standard input file.
- Numbers may be given in any standard FORTRAN format. A real may be given as an integer. Spaces in a number are considered as separators.
- Any character or set of characters which obviously does not belong to the number may be used as separator. As a consequence the text ndim = 2, is read as 2 and the text f2 = 3 as two numbers 2 and 3.

The intermediate file will be organized in blocks as follows:

Block number:	Contents
1	version number of the file
2	Global parameters for the common block CISNAS
3	Dimension parameters
4	Choice parameters
5	Parameters with respect to the coefficients
6	Parameters with respect to the linear solvers
7	Parameters with respect to the time integration
8	Parameters with respect to the initial conditions
9	Parameters with respect to the discretization
10	Parameters with respect to the multi-block process
11	Parameters with respect to the turbulence modelling
12	Parameters with respect to the compressibility
13	Parameters with respect to the output
14	Parameters with respect to the boundary conditions
15	Parameters defined by the user with respect to the special common block CUSER
16	Sequence of equations

A block may consist of one or more records. The position of the blocks as well as the parameters in the blocks is fixed. In the next sections the contents of the various blocks is described.

Block 1: version number

In order to make the file upwards compatible a version number is introduced. If the definition of the intermediate file is changed in such a way that the old files can not be interpreted immediately, a higher version number will be used.

With respect to the reading of intermediate files, there are two alternatives:

1. The (pre-)processor reads only files of the correct version number. If a file with an incorrect version number is read an error message is given. The program that translates the intermediate file into a batch file, however, will still be able to read previous version numbers. In this way it is possible to update the incorrect file by means of the batch pre-processor.
2. Both the pre-processor as the processor are able to read old versions of the intermediate file. A clear disadvantage is that these programs become more complicated.

At this moment only alternative 1 is implemented.

The present version number to be used is 3.

Block 2: Global parameters for the common block CISNAS

At this moment we define block 2 consisting of an integer and a double precision array each of length 100 in the sequence:

integer_common(100)
double_precision_common(100)

Each of the arrays starts at a new record. In the binary case each array consist of one record, in the ASCII case an array may span several records. Contents:

integer_common:

1:	IOUTPT
2:	ITIME
3-100:	0

double_precision_common:

1-100: 0d0

The positions with a zero value are meant for later use. The value 0 is essential in order to change the definition of the file without changing the version number.
All other parameters in the commons are either read from the isnas.env file, the isnas.dbg file or are initialized with standard values.

Block 3: Dimension parameters

At this moment we define block 3 consisting of an integer array IDIMENS of length 100

Contents:

1: NVIRTUAL
2: NDEGFD
3: NTRNSP
4: NTIMLV
5: NCOEFS
6: NTURB
7: NMACHINES
8-100: 0

See the remarks in block 2

Block 4: Choice parameters

At this moment we define block 4 consisting of an integer array ICHOICE of length 100

Contents:

1: IGEO
2-100: 0

See the remarks in block 2.

Array IINPUT positions 1 to 50 is filled by ISNaS itself.
The parameters NI, NJ, NK, NDIM, VLEN, MAX_POINTS, MAX_CELLS and NBLOCKS are read from the mesh input file.
The parameters MAXCON, MAXB MAXVPOINTS and Npoint_global are computed after reading the mesh and the intermediate file.

Block 5: Parameters with respect to the coefficients

Block 5 consists of NTRNSP+1 subblocks. The first subblock refers to the momentum equations, the i^{th} one (i>1) to the (i-1)-th transport equation. Each subblock starts at a new record, and the subblocks are stored in the natural sequence. For each subblock the following information is stored:

(IINCOF(i),i=1,NCOEFS), (RINCOF(i),i=1,NCOEFS)

For a description of the arrays IINCOF and RINCOF, see Section 16.10.
See also the remarks in block 2.

Block 6: Parameters with respect to the linear solvers

Block 6 consists of NTRANS+2 subblocks.
The first subblock refers to the momentum equations, the second block to the pressure equation

and the i-th one block (i>2) to the $i-2^{th}$ transport equation.
Each subblock starts at a new record, and the subblocks are stored in the natural sequence.
For each subblock the following information is stored:

(IINSOL(i),i=1,25), (RINSOL(i),i=1,25)

See the remarks in block 2

Block 7: Parameters with respect to the time integration

At this moment we define block 7 consisting of an integer array IINTIM and a double precision array RINTIM each of length 100.
Each of the arrays starts at a new record.

Contents:

IINTIM pos. 1-14 See 16.13.1.
The rest of the array must be filled with zeros.

RINTIM pos. 1-60 See 16.13.2.
The rest of the array must be filled with zeros.

See the remarks in block 2

Block 8: Parameters with respect to the initial conditions

At this moment we define block 8 consisting of an integer array IINCND and a double precision array RINCND each of length NDEGFD. Only one array is used for all blocks. Each of the arrays starts at a new record.

Contents:

IINCND pos. 1-NDEGFD See 16.12.1.
The rest of the array must be filled with zeros.

IINCND pos. 1-NDEGFD See 16.12.2.

See the remarks in block 2

Block 9: Parameters with respect to the discretization

At this moment we define block 9 consisting of an integer array IINDSC and a real array RINDSC both of length of length 100∗(NTRANS+1)

Contents:

IINDSC pos. 1-20 See 16.16.1.
RINDSC pos. 1-20 See 16.16.2.
The rest of the array must be filled with zeros.

See the remarks in block 2

Block 10: Parameters with respect to the multi-block process

At this moment we define block 10 consisting of an integer array IINBLK of length 10∗(NTRANS+3) and a double precision array RINBLK of size 10∗(NTRANS+2). Array IADMIN is filled by the

main program itself, after reading the intermediate file and array ITOPOL is filled by the subroutine reading the grid information.

Contents:

IINBLK See 16.18.4

RINBLK See 16.18.5

See the remarks in block 2

Block 11: Parameters with respect to the turbulence modelling

At this moment we define block 11 consisting of an integer array IINTUR and a real array RINTUR both of length 100. In IINTUR, positions 1-2 should be filled as defined in Section 16.14.1, positions 3-100 should be filled with zeros. In RINTUR, positions 1-14 should be filled as defined in Section 16.14.2, positions 15-100 should be filled with zeros.

Block 12: Parameters with respect to the compressibility

At this moment we define block 12 consisting of an integer array IINCOM and a real array RINCOM both of length 100. In IINCOM, positions 1 should be filled as defined in Section 16.19.1, positions 2-100 should be filled with zeros. In RINCOM, positions 1-6 should be filled as defined in Section 16.19.2, positions 7-100 should be filled with zeros.

Block 13: Parameters with respect to the output

At this moment we do not have a definition of these parameters. We suppose that an integer and a real array of length 25 each will be used. At this moment each of these arrays must be filled with 25 zeros. Both arrays should start at a new record.

Block 14: Parameters with respect to the boundary conditions

Block 14 consists of NBLOCKS subparts each referring to one subblock. Each of these parts consists of two arrays IINBC and RINBC already defined in Sections 16.11.3 and 16.11.4. Positions 1 to 2∗ndim, of IINBC, indicating the starting positions are not stored in the intermediate file. Each of these arrays starts at a new record.

See the remarks in block 2.

Block 15: Parameters with respect to the special common cuser

Block 15 is filled with the information given by the user. Contents:

Pos. 1 Number of integers for array IUSER (NIUSER)
Pos. 2 Number of reals for array USER (NUSER)

The positions 1 and 2 are stored in one record

Next record:

NIUSER integers to be stored in IUSER

Next record:

NUSER reals to be stored in USER

See the remarks in block 2

Block 16: Sequence numbers of the equations

Block 16 is filled with the sequence numbers of the equations.

A sample input file

In this section we present without any further comment an example of an input file for a very simple problem.

```
* Intermediate file for the ISNaS incompressible Navier-Stokes equations

**** BLOCK 1: version number

version  5


**** BLOCK 2: Global parameters for the common block CISNAS

* 100 integers:

IOUTPT =     0
ITIME  =     0
 0 0 0 0 0 0 0 0        # Positions  3-10 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 11-20 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 21-30 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 31-40 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 41-50 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 51-60 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 61-70 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 71-80 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 81-90 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 91-100 not yet defined



* 100 reals:

  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.    # Positions  1-10 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.    # Positions 11-20 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.    # Positions 21-30 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.    # Positions 31-40 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.    # Positions 41-50 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.    # Positions 51-60 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.    # Positions 61-70 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.    # Positions 71-80 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.    # Positions 81-90 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.    # Positions 91-100 not yet defined



**** BLOCK 3: Dimension parameters
```

```
* 100 integers:

NVIRTUAL  =     2
NDEGFD    =     3
NTRNSP    =     0
NTIMLV    =     2
NCOEFS    =     6
NTURB     =     0
NMACHINES =     1
 0 0 0                  # Positions  8-10 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 11-20 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 21-30 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 31-40 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 41-50 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 51-60 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 61-70 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 71-80 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 81-90 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 91-100 not yet defined



**** BLOCK 4: Choice parameters

* 100 integers:

IGEO =     2
 0 0 0 0 0 0 0 0 0      # Positions  2-10 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 11-20 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 21-30 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 31-40 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 41-50 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 51-60 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 61-70 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 71-80 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 81-90 not yet defined
 0 0 0 0 0 0 0 0 0 0    # Positions 91-100 not yet defined



**** BLOCK 5: Parameters with respect to the coefficients

** Subblock 5.1 momentum equations

* iincof :

    -1    -1    -1    -1    -1    -1

* rincof :

 rho    =   0.5000000D+01
 mu     =   0.5000000D+00
 fone   =   0.0000000D+00
 ftwo   =   0.0000000D+00
```

```
 fthree =   0.0000000D+00

dummies:
  0.0000000D+00

**** BLOCK 6: Parameters with respect to the linear solver

** Subblock 6.1 momentum equations



* iinsol:

                       0     # Pos. 1 is dummy
ipreco            =    2     # Preconditioner
amount_of_output  =    0
start_vector      =    2
stop              =    0
maxiter           =  200
method            =    2
file_nr           =    6
                       0     # Pos. 9 is dummy
                       0     # Pos. 10 is dummy
irestart          =   20
iorthogonal       =   10
degree of polynom =    0
numb of Ritz vals =    0
save precondit.   =    0
prec. built.      =    0
print solution    =    0
itime_one         =    1
itime_two         =    1
itime_step        =    1
 0 0 0 0 0  # Positions 21-25 not yet defined

* rinsol :

absaccuracy        =  0.0000000D+00
relaccuracy        =  0.1000000D-03
                      0.0000000D+00     # Pos. 3 is dummy
                      0.0000000D+00     # Pos. 4 is dummy
condaccuracy       =  0.0000000D+00
divaccuracy        =  0.0000000D+00
alpha              =  0.1000000D+01
  0.0000000D+00  0.0000000D+00  0.0000000D+00  0.0000000D+00  0.0000000D+00
  0.0000000D+00  0.0000000D+00  0.0000000D+00  0.0000000D+00  0.0000000D+00
  0.0000000D+00  0.0000000D+00  0.0000000D+00  0.0000000D+00  0.0000000D+00
  0.0000000D+00  0.0000000D+00  0.0000000D+00

** Subblock 6.2 pressure equation



* iinsol:
```

```
                        0     # Pos. 1 is dummy
ipreco            =   -3     # Preconditioner
amount_of_output  =    0
start_vector      =    0
stop              =    0
maxiter           =  200
method            =    2
file_nr           =    6
                        0     # Pos. 9 is dummy
                        0     # Pos. 10 is dummy
irestart          =   40
iorthogonal       =   20
degree of polynom =    0
numb of Ritz vals =    0
save precondit.   =    0
prec. built.      =    0
print solution    =    0
itime_one         =    1
itime_two         =    1
itime_step        =    1
 0 0 0 0 0  # Positions 21-25 not yet defined

* rinsol :

absaccuracy        =  0.0000000D+00
relaccuracy        =  0.1000000D-05
                      0.0000000D+00     # Pos. 3 is dummy
                      0.0000000D+00     # Pos. 4 is dummy
condaccuracy       =  0.0000000D+00
divaccuracy        =  0.0000000D+00
alpha              =  0.9750000D+00
  0.0000000D+00  0.0000000D+00  0.0000000D+00  0.0000000D+00  0.0000000D+00
  0.0000000D+00  0.0000000D+00  0.0000000D+00  0.0000000D+00  0.0000000D+00
  0.0000000D+00  0.0000000D+00  0.0000000D+00  0.0000000D+00  0.0000000D+00
  0.0000000D+00  0.0000000D+00  0.0000000D+00

**** BLOCK 7: Parameters with respect to the time integration

* iintim:

method =      1
 0 0 0 0              # Positions 2-5 not yet defined
nfrac  =     1
tvar   =     0
ndt    =     1
 0                    # Positions 9 not yet defined
istationary =      1
 0 0 0 0 0 0 0 0 0 0  # Positions 11-20 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 21-30 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 31-40 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 41-50 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 51-60 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 61-70 not yet defined
```

```
 0 0 0 0 0 0 0 0 0 0  # Positions 71-80 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 81-90 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 91-100 not yet defined

* rintim :

  0.0000000D+00                  # Pos.1 is dummy
tstart     =  0.0000000D+00
tend       =  0.2000000D+01
dt         =  0.1000000D+00
theta      =  0.1000000D+01
toutinit   =  0.2000000D+01
toutend    =  0.2000000D+01
toutstep   =  0.1000000D+00
  0.  0.  # Positions 9-10 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 11-20 not yet defined
absaccuracy_mom =  0.1000000D-03
relaccuracy_mom =  0.1000000D-03
absaccuracy_prs =  0.1000000D-03
relaccuracy_prs =  0.1000000D-03
absaccuracy_trs =  0.1000000D-03
relaccuracy_trs =  0.1000000D-03
absaccuracy_tur =  0.1000000D-03
relaccuracy_tur =  0.1000000D-03
absaccuracy_stat =  0.0000000D+00
relaccuracy_stat =  0.1000000D-01
  0.0000000D+00  0.1000000D+01  0.0000000D+00  # theta1 ... theta3
  0.0000000D+00  0.0000000D+00  0.0000000D+00  # theta4 ... theta6
  0.0000000D+00  0.0000000D+00  0.0000000D+00  # theta7 ... theta9
  0.0000000D+00                                # theta10
  0.1000000D+00  0.0000000D+00  0.0000000D+00  # dt1 ... dt3
  0.0000000D+00  0.0000000D+00  0.0000000D+00  # dt4 ... dt6
  0.0000000D+00  0.0000000D+00  0.0000000D+00  # dt7 ... dt9
  0.0000000D+00                                # dt10
  0.2000000D+01  0.0000000D+00  0.0000000D+00  # tend1 ... tend3
  0.0000000D+00  0.0000000D+00  0.0000000D+00  # tend4 ... tend6
  0.0000000D+00  0.0000000D+00  0.0000000D+00  # tend7 ... tend9
  0.0000000D+00                                # tend10
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 61-70 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 71-80 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 81-90 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 91-100 not yet defined


**** BLOCK 8: Parameters with respect to the initial conditions

* iincnd:

    -1    -1    -1


* rincnd:

  0.0000000D+00  0.0000000D+00  0.0000000D+00
```

```
**** BLOCK 9: Parameters with respect to the discretization

** Subblock 9.1 momentum equations



* iindsc:

iterm               =    1
idiscret_convection =    1
idiscret_viscosity  =    1
idiscret_pressure   =    1
iupwind             =    0
imonoton_diffusion  =    0
idiscret_production =    1
linearization       =    1
 0 0                   # Positions  9-10 not yet defined
iprint_matrix       =    0
iprint_rhsd         =    0
iprint_time_one     =    1
iprint_time_two     =    1
iprint_timestep     =    1
 0 0 0 0 0     # Positions 16-20 not yet defined
 0 0 0 0 0 0 0 0 0 0     # Positions 21-30 not yet defined
 0 0 0 0 0 0 0 0 0 0     # Positions 31-40 not yet defined
 0 0 0 0 0 0 0 0 0 0     # Positions 41-50 not yet defined
 0 0 0 0 0 0 0 0 0 0     # Positions 51-60 not yet defined
 0 0 0 0 0 0 0 0 0 0     # Positions 61-70 not yet defined
 0 0 0 0 0 0 0 0 0 0     # Positions 71-80 not yet defined
 0 0 0 0 0 0 0 0 0 0     # Positions 81-90 not yet defined
 0 0 0 0 0 0 0 0 0 0     # Positions 91-100 not yet defined



** Subblock 9.1 momentum equations



* rindsc :

parm_muscl =  0.0000000D+00
omega      =  0.0000000D+00
beta       =  0.0000000D+00
  0.  0.  0.  0.  0.  0.  0.             # Positions  4-10 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. # Positions 11-20 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. # Positions 21-30 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. # Positions 31-40 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. # Positions 41-50 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. # Positions 51-60 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. # Positions 61-70 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. # Positions 71-80 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. # Positions 81-90 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. # Positions 91-100 not yet defined
```

```
**** BLOCK 10: Parameters with respect to the multi-block process

 * iinblk:

** general information

method             =     0
subdomain_solution =     0
save precondit.    =     0
 0 0 0 0 0 0 0   # Positions  4-10 not yet defined


** Momentum equations



algorithm           =    0
maxiter             =  100
Krylov_outer        =   20
Krylov_inner        =    1
Amount_of_output    =    0
Truncation strategy =    1
 0 0 0 0

** Pressure equations



algorithm           =    0
maxiter             =  100
Krylov_outer        =   20
Krylov_inner        =    1
Amount_of_output    =    0
Truncation strategy =    1
 0 0 0 0

* rinblk :

** Momentum equations



relative accuracy    =  0.1000000D-03
relaxation parameter =  0.1000000D+01
  0.  0.  0.  0.  0.  0.  0.  0.       # Positions  8-10 not yet defined


** Pressure equations
```

```
relative accuracy    =  0.1000000D-03
relaxation parameter =  0.1000000D+01
  0.  0.  0.  0.  0.  0.  0.  0.       # Positions  8-10 not yet defined


**** BLOCK 11: Parameters with respect to the turbulence modelling

* iintur:

nturb   =     0
mturb   =     0
viscorr =     0
 0 0 0 0 0 0 0        # Positions  4-10 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 11-20 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 21-30 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 31-40 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 41-50 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 51-60 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 61-70 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 71-80 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 81-90 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 91-100 not yet defined

* rintur :

kappa      =  0.4100000D+00
E          =  0.9000000D+01
c_mu       =  0.9000000D-01
sigma_k    =  0.1000000D+01
sigma_eps  =  0.1300000D+01
ceps_one   =  0.1440000D+01
ceps_two   =  0.1920000D+01
c_D        =  0.8000000D-01
sigma_T    =  0.1000000D+01
A          =  0.2600000D+02
E_rough    =  0.3300000D+02
eta_zero   =  0.4380000D+01
betarng    =  0.1200000D-01
c_s        =  0.1000000D+00
alpha      =  0.5555556D+00
beta_star  =  0.9000000D-01
beta       =  0.7500000D-01
sigma      =  0.5000000D+00
sigma_star =  0.5000000D+00
  0.  # Position 20 not yet defined

  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 21-30 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 31-40 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 41-50 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 51-60 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 61-70 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 71-80 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 81-90 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 91-100 not yet defined
```

```
**** BLOCK 12: Parameters with respect to the compressibility

* iincom:

mcon   =     0
 0 0 0 0 0 0 0 0 0    # Positions  2-10 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 11-20 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 21-30 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 31-40 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 41-50 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 51-60 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 61-70 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 71-80 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 81-90 not yet defined
 0 0 0 0 0 0 0 0 0 0  # Positions 91-100 not yet defined

* rincom :

gamma      =  0.0000000D+00
Mach       =  0.0000000D+00
VEL_sound  =  0.0000000D+00
alpha     =  0.0000000D+00
p_v       =  0.0000000D+00
p_out     =  0.0000000D+00
  0.  0.  0.  0.                          # Positions 7-10 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 11-20 not yet defined

  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 21-30 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 31-40 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 41-50 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 51-60 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 61-70 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 71-80 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 81-90 not yet defined
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  # Positions 91-100 not yet defined


**** BLOCK 13: Parameters with respect to the output

 integer array

 0 0 0 0 0  0 0 0 0 0  # Positions  1-10 not yet defined
 0 0 0 0 0  0 0 0 0 0  # Positions 11-20 not yet defined
 0 0 0 0 0             # Positions 21-25 not yet defined

 real array

 0 0 0 0 0  0 0 0 0 0  # Positions  1-10 not yet defined
 0 0 0 0 0  0 0 0 0 0  # Positions 11-20 not yet defined
 0 0 0 0 0               # Positions 21-25 not yet defined

**** BLOCK 14: Parameters with respect to the boundary conditions
```

```
* iinbc:

# Block   1


# Boundary   1

  1        # Number of segments
    1    4        # i1, i2
    4   -1   -1   # momentum

# Boundary   2

  1        # Number of segments
    1    4        # i1, i2
    4   -1   -1   # momentum

# Boundary   3

  1        # Number of segments
    1    6        # i1, i2
   12   -1    0   # momentum

# Boundary   4

  1        # Number of segments
    1    6        # i1, i2
    6   -1   -1   # momentum

* rinbc :



# Boundary   1

  0.0000000D+00  0.0000000D+00

# Boundary   2

  0.0000000D+00  0.0000000D+00

# Boundary   3

  0.1000000D+01  0.0000000D+00

# Boundary   4

  0.0000000D+00  0.0000000D+00

**** BLOCK 15: Parameters defined by the user with respect
****           to the special common block CUSER
```

```
niuser =    0
nuser =    0


**** BLOCK 16: Sequence of the equations



* iseqeq:
    1    # momentum equation

    2    # pressure equation
```

2.2.3 File isnas.hosts

The file isnas.hosts may be provided by the user. This file is used by PVM (Parallel Virtual Machine) in order to find which computers may be used as parallel machines, i.e. as nodes or host. If the user does not give this file the standard PVM configuration stored in the file pvmmachines in the pvm directory is used. If the user has provided the file isnas.hosts in the local directory from which he runs the ISNaS program this file is used. Of course this file is only used if the parallel version of ISNaS is activated.

The file isnas.hosts has a very simple structure. It is an ASCII file containing the name of all computers that may be used in the parallel process. Each name of the computer must start at a new line and the name of the host computer must be included in this list.

An example of a file isnas.hosts is the following one:

```
dutinws
dutinwp
dutinwg
```

2.2.4 File isnas.env

The ISNaS environment file is used to define some general constants, file reference numbers, names of files and types of plotting devices. The environment file has the name isnas.env and is defined as a standard ASCII file. This file consists of comment records and records with a definition.

The standard isnas.env file can be found in the ISNaS subdirectory instal. This file can only be changed by the ISNaS installation officer. However, it is also possible to make a copy of this file and put it in your own directory. This copy may be changed according to your own personal preferences. A disadvantage is that this copy is only valid in the directory where it is available, and programs running from other directories do not read this file.
Each ISNaS program that is started checks if there is a file named isnas.env in the local directory. If this file is available the file is read and the contents are used by the program. If this file does not exist the general isnas.env file from the directory ISNaS/instal is used, where ISNaS is a substitute for the ISNaS home directory.

The isnas.env file is a standard file with comment records and data records. A comment record

starts with an asterisk, the data records start with an explanation followed by a colon (:), followed by the actual data. The isnas.env file has, a fixed sequence. Data records may not be interchanged. If this is done error messages may be the result, or more severely, the program may produce incomprehensible messages.

Contents of the isnas.env file

The present version of the isnas.env file contains the following data records (in that sequence):

name of ISNaS home directory
version number of environment file
type of operating system
name of computer
type of computer
version name of ISNaS
name of executable
number of characters in a word
approximation of infinity
machine accuracy
accuracy of a half real
Unit number for reading of ISNaS input
Unit number for writing of ISNaS output
Unit number for error messages
Unit number for ISNaS backing storage file
Unit number for temporary file
Unit number for file containing menus
Unit number for mesh file
Unit number for file sepcomp.inf
Unit number for file sepcomp.out
Unit number for intermediate input file
Default ISNaS plot file (formatted/unformatted)
file 10 (formatted/unformatted)
file 74 (formatted/unformatted)
append mode (yes/no)

Name of file 4
Name of file for menus
Name of temporary file
Name of ISNaS input file
Name of ISNaS output file
Name of (Binary mesh output file)
Name of sepcomp.inf (73)
Name of sepcomp.out (74)
Name of intermediate input file (75)
Name of ISNaS backing storage file
lenwor
Record length ISNaS backing storage file
Record length for scratch file
carriage
tem1
tem2
tem3

tem4
tem5
tem6
tem7
tem8
tem9
tem10

explanation of the data records

name of ISNaS home directory
: The complete path name of the ISNaS home directory must be given.

version number of environment file
: Do not change this number.

type of operating system
: Answers that are recognized are:
  msdos
  unix
  vax/vms

name of computer
: Host name of the computer.

type of computer
: Give type number from table below.

1	IBM
2	Cyber (NOS/VE)
3	Apollo (Unis/Aegis)
4	HP 9000 (HP/UX)
5	..... Unix
6	VAX VMS
7	Harris
8	Cray
9	IBM PC 386 + MSDOS + FTN77/386
10	CONVEX
13	Alliant
14	unknown

version name of ISNaS
: This item is generally not used.

name of executable
: This item is generally not used.

number of characters in a word
: Give an integer number.

approximation of infinity
: The largest real (or double precision) number on the computer.

machine accuracy
: The machine accuracy must be given as real number.
  For example at a 32 bits computer with double precision arithmetic: 1d-16

accuracy of a half real
: The machine accuracy for single precision reals.

Unit number for reading of ISNaS input
: if 5 is given usually standard input is meant, without file name.

Unit number for writing of ISNaS output
: if 6 is given usually standard output is meant, without file name.

Unit number for error messages
Unit number for ISNaS backing storage file
Unit number for temporary file
Unit number for file containing menus
Unit number for mesh file
Unit number for file sepcomp.inf
Unit number for file sepcomp.out
Unit number for intermediate input file
Structure of file 10 (formatted/unformatted)
    Indicates if the mesh file must be read in formatted or unformatted form.
Structure of file 74 (formatted/unformatted)
    Indicates if the sepcomp.out file must be written in formatted or unformatted form.
append mode (yes/no)
    Indicates if the output must be appended to an existing file (yes) or not (no).
    This possibility is usually not available.

Name of file 4
    This name should not be changed.
Name of file for menus
    This name should not be changed.
Name of temporary file
    If no name is given a scratch file is used.
Name of ISNaS input file
    If no name is given standard input is used.
Name of ISNaS output file
    If no name is given standard output is used.
Name of (mesh file)
Name of sepcomp.inf (73)
Name of sepcomp.out (74)
Name of intermediate input file (75)
Name of ISNaS backing storage file
lenwor
    Record length of some direct access files:
    Record length in words.
    In the open statement it is multiplied by lenwor
Record length ISNaS backing storage file
Record length for scratch file
carriage
    Information about carriage control.
    The parameter carriage indicates if the first character in a write statement
    to a file is used as carriage control parameter (yes) or not (no).
    For most unix computers no carriage control is used, however, for example
    vax/vms uses carriage control.

The parameters tem1 to tem10 are meant for future purposes. They must be set equal to 0.

sample isnas.env file for HP-UX computer

The following isnas.env file is standard imposed on HP computers running HP-UX. It gives an example of how a isnas.env file looks like.

```
*    isnas.env
```

```
*    ISNaS environment file
*
*    Extracted from the SEPRAN environment file version 2.1
*
*
*    This file contains most of the machine-dependent quantities
*    with respect to ISNaS
*    The file is read by each ISNaS program and the information is stored
*    in-core in some common blocks
*    The sequence of the information is essential
*    The file may be updated for your local computer
*    Lines starting with * are treated as comments
*    Only the information after the colon is interpreted
*
*    Special hp version for TUD faculty TWI
*
*    Start with the ISNaS home directory
*
ISNaS home directory    (SPHOME)          :/mnt/admin/isnas
version number of environment file        :1
*
*    Type of operating system
*    Recognised are unix, vms, cms, msdos, unknown, nos/ve
*
type of operating system                   :unix
name of computer                           :dutita1
*
*     Give type of computer system. Possible values:
*
*                1    IBM
*                2    Cyber   (NOS/VE)
*                3    Apollo  (Unis/Aegis)
*                4    HP 9000 (HP/UX)
*                5    .....    Unix
*                6    VAX      VMS
*                7    Harris
*                8    Cray
*                9    IBM PC 386   MSDOS + FTN77/386
*               10    CONVEX
*               13    Alliant
*               14    unknown
*
type of computer (number from list)        :4
version name of ISNaS                      :
name of executable                         :
*
*    Some machine dependent constants
*
number of characters in a word             :   4
approximation of infinity                  :1e77
machine accuracy                           :1e-15
accuracy of a half real                    :1e-6
*
*  Unit numbers for files
```

```
*
Unit number for reading of ISNaS input    :  5
Unit number for writing of ISNaS output   :  6
Unit number for error messages            :  4
Unit number for ISNaS backing storage file:  2
Unit number for temporary file            :  3
Unit number for file containing menus     :  17
Unit number for mesh output file          :  10
Unit number for file sepcomp.inf          :  73
Unit number for file sepcomp.out          :  74
Unit number for intermediate input file   :  75
*
*
*  Indication whether the files 10 and 74 are binary (unformatted) or formatted
*  Recognised are formatted and unformatted
*
file 10                                   :formatted
file 74                                   :unformatted
*
*  Indication whether the output file must be appended to an existing output
*  file (yes) or not (no)
*
append mode                               :no
*
*  Standard and default names for files:
*  If no name is given the file is supposed to be without name
*  and only the unit number is used in the open statement
*  The files 5 and 6 are not opened
*  If the reading and writing ISNaS files have other unit numbers than
*  5 and 6, they must have a name
*  The temporary file may have no name in which case it is opened as
*  a scratch file or it mat have the name
*  tmp
*  In the last case (only meant for UNIX computers) the temporary file is
*  opened as a temporary file at the directory /tmp
*
file 4                                    :SPHOME/bin/errornewmsg
file for menus                            :SPHOME/bin/menumsg
temporary file                            :
ISNaS input file                          :
ISNaS output file                         :
(Binary mesh output file)                 :finvol.new
sepcomp.inf (73)                          :sepcomp.inf
sepcomp.out (74)                          :sepcomp.out
intermediate input file (75)              :isnasinp.cmp
ISNaS backing storage file                :isnasback
*
*  Record length of some direct access files:
*  Record length in words
*  In the open statement it is multiplied by lenwor
*
lenwor                                    :4
Record length ISNaS backing storage file  :1157
Record length for scratch file            :1024
```

```
*
* Information about carriage control
* The parameter carriage indicates if the first character in a write statement
* to a file is used as carriage control parameter (yes) or not (no)
* For most unix computers no carriage control is used, however, for example
* vax/vms uses carriage control
*
carriage                                    :no
*
*  The next 10 positions are reserved for later use:
*
tem1                                        :0
tem2                                        :0
tem3                                        :0
tem4                                        :0
tem5                                        :0
tem6                                        :0
tem7                                        :0
tem8                                        :0
tem9                                        :0
tem10                                       :0
```

2.2.5 File isnas.dbg

Besides the usual facilities provided by compilers, etc., ISNaS offers an extra debug possibility which is especially meant for the checking of the buffer array and the memory management.
If the user wants to activate the debug facility he has to provide a file named `isnas.dbg` in the directory where he is executing the ISNaS job.

In the file `isnas.dbg` debug commands may be given. For these commands the rules given in Section 2.2.3 are valid. The following commands are recognized:

```
debug_level = d
nbuffr = n
int_infin = i
print_layout = i
print_sequence = i
```

Meaning of these parameters:

debug_level = d defines the debug level.
The following values of d may be used:

d=0 no debugging is activated.

d=1 lowest level of debugging. Each array in the buffer is surrounded by two control integers. In each call of a memory management subroutine these control integers are checked. In this way simple violations of array bounds may be detected.

d=2 next level of debugging. Except the checks introduced in $d = 1$ also the buffer array is set equal to int_infin. If space is freed in the buffer array, again this part is set equal to int_infin. In this way incorrect initializations may be detected. Default value: $d = 0$.

nbuffr = n redefines the value of $nbuffr$, i.e. the buffer length. However, it is with this statement not possible to extend the buffer length only to decrease the part actually used.
The use of this statement makes sense in combination with *debug_level* 2, since then a smaller part of the buffer array is used and initialized. Especially if the buffer array is larger than

the available memory, swapping to disk may be avoided by decreasing `nbuffr`. For example on a pc the value $n = 500000$ decreasing `nbuffr` from 5000000 to 500000 is recommended for efficiency reasons.
Default value: n set by the computer installation.

int_infin = i defines the value of the largest integer. This value is only used if $d = 2$ to initialize the buffer.
Default value: $2^{31} - 1$.

print_layout = i defines the type of layout to be used for printing matrices. The following values are available:

1: Each matrix, right-hand side and solution is printed in the standard way, i.e. each i,j,k index starts at a new line. The layout per index is index followed by all entries of corresponding to this index If all entries corresponding to the index are 0, the line is skipped

2: See 1, however, with respect to the matrices only non-zero entries are printed. Each matrix entry is provided with component sequence number and offset.

print_sequence = i defines the sequence to be used for printing. The following values are available:

1. The standard sequence is used, i.e. first index i, second j and third one k
2. The sequence j, k, i is used.
3. The sequence k, i, j is used.
4. The sequence Inv(i), j, k is used. This means that i runs from top to bottom and that the value of i is reversed in the following sense:
   i=1-nvirtual becomes i=ni+1+nvirtual,
   i=2-nvirtual becomes i=ni+nvirtual,
   i=3-nvirtual becomes i=ni-1+nvirtual and so on
   This possibility has only been implemented for 2D
5. The sequence i, Inv(j), k is used. This means that j runs from top to bottom and that the value of j is reversed in the following sense:
   j=1-nvirtual becomes j=nj+1+nvirtual,
   j=2-nvirtual becomes j=nj+nvirtual,
   j=3-nvirtual becomes j=nj-1+nvirtual and so on
   This possibility has only been implemented for 2D

Remark:

Debugging may influence the performance of the computations in a negative way and should only be used in a developing stage or if a program is tested or not trusted.

Combination with `set output_level=3` in the input file may extend the information available and hence make it easier to detect errors.

Chapter 3

Output files

3.1 Introduction

In this chapter we describe the various output files that are produced by the ISNaS incompressible program.

It concerns the following files:

isnasinp.cmp 2.2.2	Contains the intermediate file between pre-processor and computational program. This file has already been treated in 2.2.2 There is no need to keep this file for output purposes.
isnascomp.out 3.2.1	Contains the computed solution at the time-levels defined in the input.
isnasback 3.2.2	Restart file. This file contains the data for restarts.

The names of the files isnasback, isnasinp.cmp and isnascomp.out are given in the file isnas.env. These names may be changed by using a local file isnas.env or by changing the global file isnas.env.

3.2 Description of some of the output files

In this section we give a complete description of the files: isnascomp.out and isnasback.

3.2.1 File isnascomp.out

This section describes the structure of the file isnascomp.out. This file contains the output of all solutions at all time-levels where the user has requested that output must be performed. The contents of this file can only be interpreted in connection with the file finvol.new.

`isnascomp.out` may be both formatted and unformatted. Whether the file is considered formatted or unformatted depends on the contents of the file `isnas.env`. See 2.2.4

The format for the isnascomp.out file is fixed so that a read program will have no problem interpreting the data.

The general structure of the isnascomp.out file is:

For each time level

    For each unknown and special quantity
        Name of unknown

time, sequence number, type number

For each block
block sequence number, ni, nj, nk
For each component of the unknown and special quantity
Solution in all non-virtual vertices of the block.
The solution is printed in the sequence (1,1,1), (2,1,1), ... (ni,1,1), (1,2,1), ...

The sequence number indicates the sequence number of the unknown or special quantity. This sequence is analogous to the the storage in ISNaS as described in Section 16.2, but not entirely. The sequence of these quantities is defined as follows:

Laminar incompressible flow 1 Velocity (NDIM Cartesian components)

2 Pressure

3 Stream function

4-3+NTRNSP Transport quantities

Turbulent incompressible flow 1 Velocity (NDIM Cartesian components)

2 Pressure

3 Stream function

4-3+NTRNSP Transport quantities

4+NTRNSP-3+NTRNSP+NTURB Turbulence unknowns, see 16.2

4+NTRNSP+NTURB turbulent viscosity

5+NTRNSP+NTURB turbulent intensity

6+NTRNSP+NTURB turbulent length scale

7+NTRNSP+NTURB Reynolds stresses (symmetric tensor with 3 (NDIM=2) or 6 (NDIM=3) components)

Laminar compressible flow

1 Velocity (NDIM Cartesian components)

2 Pressure

3 Stream function

4 Enthalpy h

5 Density ρ

6 Mach number

Name of unknown is a 30 character string containing the name of the variable to be outputed.
time is the actual output time
The type number indicates what type of quantity is printed. Possible values:

1 Scalar quantity

2 Vector quantity with NDIM components

3 Symmetric tensor quantity with 3 or 6 components

The format that is used in case of a formatted file sepcomp.out is the following:

```
For each time-step at which output is written:
   Record 1:  ndim, nblocks, text                          format:  i3,i5,12x,a
   For each quantity to be written:
      Record 1:  name                                      format:  a
      Record 2:  time, sequence_number, type_number, text
                                                           format:  f12.5,2i3,12x,a
      For each block:
         Record 1:    iblock, ni, nj, nk, text             format:  4i8,12x,a
         For each component of the quantity:
            Solution                                       format:  5d16.8
```

3.2.2 File isnasback

Since a restarted computation may itself want to write a restart file, it must be able to contain the results of more than one session at the same time. With direct access storage, this can be achieved because such files can be accessed in a random order with read and write instructions.

Another advantage of a direct access file is that we can write information at the beginning of the file after the computation has finished. For instance, the number of written solutions is in general not known beforehand since computation may terminate before reaching the end time by either user interrupt of failure to converge in the linear solver.

Direct access files are divided into records of fixed length. The restart file contains a list of sessions. Each session consists of a list of time steps. The first record of the restart file contains information about the list of sessions and about the division of the restart file into records.

```
RECORD 1.

RECLEN     -   Length of a record
IFREE      -   The record number of the first free record on the file.
MAXNSESS   -   The maximum number of sessions stored in the file.
NSESSIONS  -   The number of ISNaS sessions stored in the file.

CHARACTER*20 SESNAME(1:MAXNSESS) - for 1 <= i <= NSESSIONS, SESNAME contains
               the name of session number i.

INTEGER      SESRECORD(1:MAXNSESS) - for 1 <= i <= NSESSIONS, SESRECORD(i)
               contains the first record number of session i.
```

Each session itself may contain the results of more than one time step. Therefore, the information to be stored must be divided into the parts dependent and independent on the time step. The part dependent on the time step is stored as a list of time steps. The part independent on the time step consists of NDIM, NTRNSP, NTURB, and ITOPOL .

```
RECORD ISESSION

ISESSION  # record number of the next session stored in the restart file.
          # ISESSION=IFREE if there are no more sessions stored.
NDIM      # dimension of the problem
NTRNSP    # no. of transport equations
NTURB     # no. of turbulence equations
ITIME     # record number of the first time step stored.
ITOPOL    # block topology
```

The actual information is stored in the records following this general information.

```
RECORD ITIME

TIME       # time at which the solution was computed
ITIMENEXT  # record number where the next time step is stored
           # ITIME = -1 if it is the last time step
ISOL       # record number where the first part of the solution is stored
```

In summary:

```
Restart_file = list of sessions

session = list of time steps

time step = list of the solution

The advantage of this list structure is that it is very easy to
append information to it.
```

Chapter 4

The structure of the main program

4.1 Introduction

In this chapter the structure of the main programs of the incompressible code is described. In fact in a parallel environment we use two main programs: the host program running at the host only and the node programs running at all nodes and possibly also at the host. The host program has complete control over the process, the node programs are just slaves solving one iteration and one time-step for a number of blocks only. In Section 4.2 the structure of the host program is described, Section 4.3 outlines the node programs. The starting and closing subroutines for host and node are described in the sections 4.4.1, 4.4.2 and 4.4.3.

In this chapter we describe the following subroutines:

ishstart 4.4.1	Starting subroutine at the host
isnstart 4.4.2	Starting subroutine at the nodes
isfinish 4.4.3	Closing subroutine at the nodes and host
isinifil 4.4.6	Open files
isinitcb 4.4.4	Initialize isnas machine-independent common blocks
isinitmd 4.4.5	Set machine-dependent quantities
isacth 4.4.7	Performs different actions at node or host.

4.2 Structure of the host program

4.2.1 Subroutine isnmainh

4.3 Structure of the node program

4.4 Detailed description of some general starting and closing subroutines

4.4.1 Subroutine ishstart

Description

This subroutine starts the process at the host. It should be called as first subroutine in the main host program.
The starting subroutine has the following tasks:

- Initialization of all machine-independent common blocks.
- Initialization of all machine-dependent common blocks. Information with respect to the machine-dependent quantities is given in the file isnas.env, which is stored in the main directory isnas/instal.
- Opening of the standard input and output files, if these files correspond to named files
- Initializing the timer to zero
- Opening of the error message file
- Opening of the scratch file for swapping purposes.
- Starting the memory manager.
- Starting the parallel process if necessary.

Heading

```
subroutine ishstart ( nbuffr )
```

Parameters

integer nbuffr

nbuffr declared length of array ibuffr

Input

nbuffr must have a value.
Array ibuffr 5.5 must have been dimensioned with length nbuffr.

Output

Array ibuffr has been initialized. The common blocks have been initialized. The timer is set equal to zero. Some files have been opened.

Subroutines called

ishstart calls some specific subroutines:

isiniclk 8.2.1	initialize the clock
isinifil 4.4.6	open files
isinitcb 4.4.4	initialize isnas machine-independent common blocks
isinitmd 4.4.5	set machine-dependent quantities
isstarmm 5.6.1	start memory manager

4.4.2 Subroutine isnstart

Description

This subroutine starts the process at the node. It should be called as first subroutine in the main node program.
Initialize machine-independent commons.
Set timer equal to zero.
Open scratch file for swapping.
Start the memory manager.
Start the parallel process.
Receive start information from host. It concerns the machine-dependent commons cisnas, cisnasch, cfile3 and and the arrays iinput, rinput, coor, solut, itopol and iadmin.

Heading

```
subroutine isnstart( nbuffr , isiinput, isrinput, iscoor,
                     issolut, isitopol, iadmin,   nblocks )
```

Parameters

integer iscoor(*), iadmin(*), isiinput(*), isitopol, isrinput(*), issolut(*), nblocks , nbuffr

nbuffr declared length of array ibuffr

isiinput Integer array of length nblocks containing the sequence numbers of the various integer input arrays iinput (see 16.9) with respect to the various blocks. Only the first position and all positions corresponding the blocks present at the node are filled. All other ones are set equal to zero.

isrinput Integer array of length nblocks containing the sequence numbers of the various double precision input arrays rinput (see 16.9) with respect to the various blocks. Only the first position and all positions corresponding the blocks present at the node are filled. All other ones are set equal to zero.

iscoor Integer array of length nblocks containing the sequence numbers of the various co-ordinate arrays coor (see 16.3) with respect to the various blocks. Only the positions corresponding the blocks present at the node are filled. All other ones are set equal to zero.

issolut Integer array of length nblocks containing the sequence numbers of the various solution arrays solut (see 16.4) with respect to the various blocks. Only the positions corresponding the blocks present at the node are filled. All other ones are set equal to zero. The arrays solut contain the initial solution vectors.

isitopol Integer array of length nblocks containing the sequence numbers of the various topology arrays itopol (see 16.18.2) with respect to the various blocks. Only the positions corresponding the blocks present at the node are filled. All other ones are set equal to zero.

iadmin Integer array of length nblocks containing information about the blocks stored at the node (see 16.18.1)

nblocks Number of blocks in the grid.

Input

nbuffr must have a value
Array ibuffr 5.5 must have been dimensioned with length nbuffr

Output

Array ibuffr has been initialized.
The machine-independent common blocks have been initialized.
The timer is set equal to zero.
Some files have been opened.
nblocks has got a value.
The arrays isiinput, isrinput, iscoor, issolut, isitopol and iadmin have been filled, as well as their corresponding arrays in IBUFFR (except of course for iadmin).

Subroutines called

isnstart calls some specific subroutines:

isiniclk 8.2.1	initialize the clock
isinifil 4.4.6	Open files
isinitcb 4.4.4	Initialize ISNaS machine-independent common blocks
isstarmm 5.6.1	Start memory manager

4.4.3 Subroutine isfinish

Description

Standard closing subroutine both at host and node. isfinish has the following tasks:

- Closing of all files that have been opened before.
- Sending the last information from node to host. At this moment this is for example the array with timing information.
- Stopping the parallel program.
- Stopping the node program.

Heading

```
call isfinish
```

Output

All files have been closed.
All necessary information has been sent from node to host.

4.4.4 Subroutine isinitcb

Description

This subroutine initializes all machine-independent common blocks.

Heading

```
subroutine isinitcb
```

Output

All machine-independent commons have been initialized.

4.4.5 Subroutine isinitmd

Description

Stores machine-dependent quantities in common blocks. Input about these quantities is read in the file isnas.env, which may be a local file. If this local file does not exist, the global isnas.env file is used.

Heading

```
subroutine isinitmd
```

Output

The machine-dependent commons have been filled.

4.4.6 Subroutine isinifil

Description

This subroutine opens all files (except for standard input and output if these files have unit number 5 and 6, or have no name). Which files are opened depends on the parameter ifile.
By a repeated number of calls more files may be opened.

The subroutine is machine dependent.

Heading

```
subroutine isinifil ( ifile )
```

Parameters

integer ifile

ifile Formal file number of file to be opened. In many practical situations this is also the actual file number.
Possible values for ifile are:

2: Standard backing storage file (iref2), reset information

-2: Standard backing storage file (iref2), old file

3: Standard scratch file (iref3)

4: Standard error message file (irefer)

5: Standard input file

6: Standard output file

10: File containing the grid (iref10)

73: File containing information of the problem (iref73) (sepcomp.inf)

74: File containing the solutions (iref74) (sepcomp.out)

75: Intermediate input file (iref75) (isnasinp.cmp)

With respect to the files 10, 73 and 74 a negative value of ifile denotes that the file must already exist. If the file does not exist an error message is given. If ierror = -1, ierror is set to 1 and control is given back to the calling subroutine , otherwise the program is stopped With respect to file 4 a positive value of ifile denotes that the file must already exist and is not changed, a negative value means that the file may be new.

Input

ifile must have a value.

Output

The file has been opened.

4.4.7 Subroutine isacth

4.5 Organization of the multi block process in a parallel environment

With respect to the parallel multiblock process, we have the following goals:

- We have a separate node and host program. The basic part of the node program is formed by a node subroutine, which also may be called by the host in order to solve some blocks locally.

- Functional decomposition

- Hiding of tasks for the programmer as much as possible (object oriented programming)

We assume that we have a host program at this moment called ishostexe and a node program isnodexe. The main body of the node program is formed by the subroutine isnode, which is also called by ishostexe.

With respect to the node and the host buffer it has been decided that the node program (isnodexe) and the host program (ishostexe) each have their own buffer. However, the subroutine isnode uses the buffer from the calling program. That means that with respect to the one-block solver at the host the host buffer is utilized.

The next schemes describe the functionality of the parallel multiblock algorithm in terms of send and receive commands. In fact the complex algorithm that is used is very much simplified and only the aspects with respect to sending and receiving have been sketched. In these schemes the commands send and receive mean actually send and or receive if data is transported from host to a node. In case of transport of data within the host from main ishostexe to isnode or vice versa, send and receiving may mean copying of data or alternatively setting of pointers whichever is the most suitable.

Functionality of the host program ishostexe

Start: send all start information for all blocks to the nodes.
internally this means that a loop over all blocks is carried out in this part.

For all t do

For all equations do

While not converged do

Send information with respect to all boundaries of all blocks from host to nodes.
Send also extra necessary information like time t, type of equation to be solved etcetera.

For all blocks at host do

Solve one block by subroutine isnode, including send result.
Solving implies building matrix and right-hand side and solution of linear system.

receive information of all boundaries from all blocks at other nodes.

Carry out the multiblock algorithm and check convergence.

At some time-levels

Receive information of complete vector fields for post-processing purposes.

Finish: send a message to all nodes that the process must stop and — receive all final informations

Functionality of the node program isnodeexe

Start: receive start information for all blocks from the host. Internally this means that a loop over all blocks is carried out in this part.

While not ready do

Receive all information of the boundaries including the extra information for all local blocks.
This information may also include a message that ready = true.

For all local blocks do

Solve one block by subroutine isnode including send

At predefined levels do

Send complete vector field to host.

Finish the process, send final information to host and stop.

Remarks

In this pseudo code it has been assumed that isnode only solves a series of blocks sequentially for one equation and one time-step. It may be possible that for some computations like for example the computation of v* in the pressure- correction formulation, in fact more steps must be carried out in one iteration. We do not exclude that possibility, but we shall consider this point in the actual implementation of the various algorithms.

subroutine isnode carries out two types of functionalities:

- Solve a set of single blocks
- Send boundary information to other blocks

The subroutines to send and receive boundary information must contain some intelligence. In fact these subroutines must decide themselves:

- If information must be copied or received
- If the actual send can be performed

For that purpose this subroutine needs a topology table and a so-called node/block table.
From the previous schemes it is clear that the following subroutines are required:

isnode **??**	Main subroutine
issenstr **??**	Send start information
isrecstr **??**	Receive start information
isfinihs **??**	Finish at host
isfinind **??**	Finish at node
issnddhs **??**	Send boundary and extra information from host to node This extra information must also contain messages of the type that a process must end.
isrcddhs **??**	Receive boundary information from node
isrcddnd **??**	Receive boundary information from host
issnddnd **??**	Send boundary information to host

and the main programs:

ishostexe and isnodeexe

Chapter 5

Memory management

5.1 Introduction

In this chapter the ISNaS memory management subroutines are treated.
The memory management is based on the SEPRAN memory management with some adaptations to prevent the use of common blocks. The memory management system is completely designed for the use in FORTRAN 77, however, it will be still applicable in FORTRAN 90. It may be expected that once ISNaS is translated in FORTRAN 90, some of the concepts in the memory management system will not longer be used. Especially the creation of temporary space will be performed by allocate and de-allocate. In Section 5.2 the tasks of the memory management system will be described.
In Section 5.3 an overview of the memory management tools is given.
Section 5.4 shows how to use the memory management system.
Section 5.5 describes the structure of the buffer array IBUFFR and finally in Section 5.6 the various tools are treated.

5.2 Tasks of the memory management system

The memory management system stores arrays in a large buffer array. It has been decided to use an integer buffer array (ibuffr 5.5) for this purpose. As a consequence double precision numbers are stored in two integer positions. In order to prevent problems with the organization of memory in computers, double precisions must always start at an odd position.
In order to prevent unwanted messages of FORTRAN checkers like FORCHECK it has been decided to use an equivalent double precision buffer array BUFFER. This array BUFFER uses exactly the same positions as IBUFFR, since the first positions of both arrays are equivalenced. Since FORTRAN 90 still contains the equivalence statement, one may expect that this option is available for the next 15 years.
If a new array is created it is first checked if there is enough space at the end of ibuffr. If so, the array is put at the end. Otherwise it is checked if there is a gap in ibuffr where the new array can be stored. If there is no gap that is large enough it is tried to create space by compression, i.e. moving arrays to the start of ibuffr and removing of free space. If even then not enough contiguous space can be found, the local swapping mechanism is started and a warning is given. In the local swapping mechanism all arrays with priority less than 10 may be written to backing storage. Newly created or activated arrays automatically get the priority 10. If an array is not allowed to be swapped it should get a priority 100, which means that the priority will never be decreased. If even the swapping mechanism is not able to create enough space, the process is stopped with an error message. It is clear that the swapping mechanism can only work if the priority is decreased explicitly. To that end all priorities, except those with priority number 100 are decreased at the end of each main subroutine by an explicit call to a specific tool. To address

a specific array in the buffer, the buffer is addressed with a pointer. Each array created in the buffer gets a so-called sequence number which should be stored in an integer. The first part of the buffer array contains an administration which keeps the starting addresses of the arrays, checks whether these arrays are in-core and contains priority numbers.

From this structure it is clear that each time an array is activated or an array is created, all pointers may get different values. As a consequence first all arrays must be created or activated and then all pointers should be retrieved. If a new activation takes place all pointers should be recomputed. After each call of a main subroutine the priority of all arrays in ibuffr should be decreased in order to be able to use the swapping mechanism. Furthermore arrays that are no longer in use, should be removed immediately in order to prevent unnecessary garbage collection or swapping. For example if a matrix is created, this matrix should be removed after the linear solver has been applied. Temporary arrays should always be constructed after the activation or creation of the permanent arrays. The reason is that for temporary arrays only a pointer is returned and no reactivation is possible. A new call of a mm subroutine destroys the temporary space.

5.3 An overview of the memory management tools

The following memory management tools are available:

Tool	Description
isstarmm 5.6.1	Initialize memory management for one buffer array
isactive 5.6.2	Activate old permanent array
isicreat 5.6.3	Create or activate new permanent integer array
isdcreat 5.6.4	Create or activate new permanent double precision array
isiexten 5.6.5	Extend a permanent integer array
isdexten 5.6.6	Extend a permanent double precision array
istmspac 5.6.7	Create space for a temporary array
isdelarr 5.6.8	Delete permanent array
isfreesp 5.6.9	Check how much contiguous space is available in ibuffr
isendspc 5.6.10	reserve space at the end of the buffer and decrease active length temporarily
islreset 5.6.11	Reset buffer length to old value, i.e. remove array created by isfreesp
isdcrpri 5.6.12	Decrease priority
issetpri 5.6.13	Set priority of an array explicitly
isingetp 5.6.14	Get pointer of an array in IBUFFR
isdpgetp 5.6.15	Get pointer of an array in BUFFER
isinleng 5.6.16	Get length of an array, expressed in integers
isdpleng 5.6.17	Get length of an array, expressed in reals

5.4 How to use the memory management system

In order to use the mm-system it is necessary to declare a large integer buffer array ibuffr 5.5 in blank common. Next ibuffr must be initialized using subroutine isstarmm. New permanent or temporary arrays may be created in main ISNaS subroutines only. Deletion and activating of arrays may also only take place at the main level. Each array is represented by two integers ISarray and iparray, where array is the name of the array (at most 6 characters). ISarray contains the sequence number of the array with respect to the internal administration of ibuffr. ISarray should be initialized to zero the first time the array is created. iparray contains the pointer referring to the starting positions of the array in ibuffr. This pointer should be recomputed each time memory management subroutines are called. In the main ISNaS subroutine the integer array "array" is addressed by ibuffr(iparray), in the actual ISNaS subroutine the array gets its usual array name and corresponding declaration, i.e. real, double precision or integer.

So a typical main ISNaS main program and subroutines may look like:

```
      program isnas_main
      implicit none
      integer ibuffr, nbuffr
      parameter ( nbuffr=1000000 )
      common ibuffr(nbuffr)
      call isnas_main_sub ( nbuffr )
      end

      subroutine isnas_main_sub ( nbuffr )
      implicit none
      integer nbuffr

      integer nvector
      parameter ( nvector=100 )

c     --- Start memory manager

      call isstarmm ( nbuffr, nvector )

c     --- Subroutines to create and manipulate arrays
              .
              .
              .
              .

c     --- Call specific main subroutine

      call main_example ( isar1, isar2 )
              .
              .
              .

      end


      subroutine main_example ( isar1, isar2 )
      implicit none
      integer isar1, isar2

      integer ibuffr
      double precision buffer(1)
      common ibuffr(1)
      equivalence ( ibuffr(1), buffer(1) )

      integer ipar1, ipar2, lenar2, lentmp, iptmp

c     --- activate array iar1, create array ar2 and reserve temporary space
c         the temporary space must be created as last array

      call isactive ( isar1 )
      isar2 = 0
      lenar2 = ....
      call isdcreat ( isar2, lenar2 )
      lentmp = ....
```

```
      call istmspac ( lentmp, iptmp )

c     --- Get pointers, except from temporary array

      ipiar1 = isingetp ( isar1 )
      ipiar2 = isdpgetp ( isar2 )

c     --- Call actual subroutine

      call subr1 ( ibuffr(ipar1), buffer(ipar2), ibuffr(iptmp) )

c     --- decrease priorities

      call isdcrpri

      end
```

5.5 The structure of array ibuffr

Array ibuffr consists of three parts: the administration, the actual arrays and some space needed for sorting. The global structure is:

administration	actual arrays	work space

The administration is filled as follows: Positions 1-10: general information
Positions 11-10+3*nvector information of the separate arrays

1 nbuffr Declared length of ibuffr.

2 nvector Maximum number of arrays that may be stored in ibuffr.

3 lastused Last array sequence number in use.

4 firstfree Next free position in ibuffr.

5 lastpos Last position ever used.

6 ibufst First free position to be used in ibuffr.

7 nbufeff Effective value of nbuffr, i.e. nbuffr-6*nvector.

8

9

10

11 - 10+3*nvector may be considered as a two-dimensional array infor(1:3,1:nvector), where infor(i,j) corresponds to ibuffr(10+i+(j-1)*3)

infor(1,j) contains the starting address of array j, where j denotes the sequence number of the array.
If the array is swapped to backing storage infor(1,j) gets a negative value referring to the record from which the array is stored at backing storage.

infor(2,j) contains the length of array j in integers.
Each double precision takes two positions.

infor(3,j) contains the priority number of array j.

11+3*nvector - nbuffr-6*nvector Actual arrays.

nbuffr-6*nvector+1 - nbuffr These positions are used by the mm subroutines for sorting and other information.
Positions nbuffeff+1 - nbuffeff+nvector correspond to the array inback as defined in the SEPRAN common carbac.

Positions nbuffeff+nvector+1 - nbuffeff+2*nvector correspond to the array lenbac as defined in the SEPRAN common carbac Positions nbuffeff+2*nvector+1 - nbuffeff+6*nvector are used for sorting.

5.6 Detailed description of the memory management tools

5.6.1 Subroutine isstarmm

Description

Initialize memory management for one buffer array.
The administration is set and if debug_level = 2, the array is set to infinity.

Heading

```
subroutine isstarmm ( nbuffr, nvector )
```

Parameters

integer nbuffr, nvector

nbuffr declared length of array ibuffr.

nvector maximal number of arrays that may be stored in ibuffr.

Input

nbuffr and nvector must have a value

Output

Array ibuffr 5.5 has been initialized

5.6.2 Subroutine isactive

Description

Activate old permanent array.
If the array is in-core only the priority is reset to 10.
If the array is at backing-storage also the array is reread into ibuffr.

Heading

```
subroutine isactive ( name )
```

Parameters

integer name

name integer variable containing the sequence number of the array to be activated.

Input

name must have a value Array ibuffr must have been filled before

Output

Array ibuffr 5.5 has been changed

5.6.3 Subroutine isicreat

Description

Create space or activate new permanent integer array The priority is set to 10

Heading

`subroutine isicreat ( name, length, namearray )`

Parameters

integer name, length

character ∗(∗) namearray

length length of the array to be created in integers

name Integer variable containing the sequence number of the array to be created. name must be either 0 or have the value of the array to be reactivated.

namearray String containing the name of the array to be created. This name is used in error messages.

Input

name, namearray and length must have a value.
Array ibuffr 5.5 must have been filled before.

Output

Array ibuffr has been changed

5.6.4 Subroutine isdcreat

Description

Create space or activate new permanent double precision (real) array The priority is set to 10

Heading

`subroutine isdcreat ( name, length, namearray )`

Parameters

integer name, length

character ∗(∗) namearray

length length of the array to be created in double precision reals

name Integer variable containing the sequence number of the array to be created. name must be either 0 or have the value of the array to be reactivated.

namearray String containing the name of the array to be created. This name is used in error messages.

Input

name, namearray and length must have a value.
Array ibuffr 5.5 must have been filled before.

Output

Array ibuffr has been changed

5.6.5 Subroutine isiexten

Description

Extend the length of an existing permanent integer array. The priority is set to 10.

Heading

```
subroutine isiexten ( name, length )
```

Parameters

integer name, length

length extra length of the array to be extended in integers

name Integer variable containing the sequence number of the array to be extended. name must have the value of the array to be extended.

Input

name and length must have a value.
Array ibuffr 5.5 must have been filled before.

Output

Array ibuffr has been changed

5.6.6 Subroutine isdexten

Description

Extend the length of an existing permanent double precision array. The priority is set to 10.

Heading

```
subroutine isdexten ( name, length )
```

Parameters

integer name, length

length extra length of the array to be extended in double precisions.

name Integer variable containing the sequence number of the array to be extended. name must have the value of the array to be extended.

Input

name and length must have a value.
Array ibuffr 5.5 must have been filled before.

Output

Array ibuffr has been changed

5.6.7 Subroutine istmspac

Description

Create space for temporary array.
subroutine istmspac must be called as last of the series of memory management creation or activation subroutines. It sets a pointer, but does not actually set anything in ibuffr, except for debugging purposes. As a consequence a new call of a memory management creation or activation subroutine, including istmspac itself destroys the temporary space. So in one call of istmspac all temporary space. must be created. This means that this space includes both real and integer space. For double precisions the length in reals must be multiplied by the parameter intlen as stored in common cisnas in order to get the length in integers.

Heading

```
subroutine istmspac ( length, ipstar )
```

Parameters

integer length, ipstar

length length of the array to be created in integers.

ipstar starting address of the work space.

Input

length must have a value.
Array ibuffr must have been filled before.

Output

Array ibuffr 5.5 has been changed.
ipstar has got a value.

Remarks

Once ipstar has got a value the starting position of all subarrays created by this one call may be computed. Since double precision reals should always start at an odd position it is good practice to define first all double precision reals and then the integers. This avoids checking of starting addresses.

5.6.8 Subroutine isdelarr

Description

Delete permanent array

Heading

```
subroutine isdelarr ( name )
```

Parameters

integer name

name integer variable containing the sequence number of the array to be deleted.
name is reset to 0 at output

Input

name must have a value.
Array ibuffr must have been filled before.

Output

Array ibuffr 5.5 has been changed

5.6.9 Subroutine isfreesp

Description

Check how much contiguous space is available in ibuffr

Heading

```
subroutine isfreesp ( ipstar, maxlen )
```

Parameters

integer ipstar, maxlen

ipstar starting address from which the free space is measured.

maxlen number of free places in ibuffr from ipstar (in integers).

Input

ipstar must have a value.
Array ibuffr 5.5 must have been filled before.

Output

maxlen has got a value.

5.6.10 Subroutine isendspc

Description

reserve space at the end of the buffer and decrease active length temporarily.
This subroutine is used to reserve a part of ibuffr 5.5 which should not be effected by the mm-subroutines. In fact the actual length of ibuffr is decreased. This is necessary if in a subroutine using mm subroutines ibuffr is used in the call

Heading

```
subroutine isendspc ( ipstar, length, nbufol )
```

Parameters

integer length, ipstar, nbufol

length length of the array to be created in integers.

ipstar starting address of the work space.

nbufol value of nbuffr at input. This value may be used to reset nbuffr by a call to islreset.

Input

length must have a value.
Array ibuffr must have been filled before.

Output

Array ibuffr has been changed.
ipstar and nbufol have got a value.

5.6.11 Subroutine islreset

Description

Reset buffer length to old value, i.e. remove array created by isfreesp

Heading

```
subroutine islreset ( nbufol )
```

Parameters

integer nbufol

nbufol value to which nbuffr must be reset.
Output of subroutine isendspc

Input

nbufol must have a value.
Array ibuffr 5.5 must have been filled before.

Output

Array ibuffr has been changed

5.6.12 Subroutine isdcrpri

Description

decrease the priorities of all arrays stored in ibuffr 5.5 except those with priority number 100

Heading

```
subroutine isdcrpri
```

Input

Array ibuffr must have been filled before.

Output

Array ibuffr has been changed.

5.6.13 Subroutine issetpri

Description

Set the priority of one array stored in ibuffr 5.5 explicitly. This is especially meant to give an array priority number 100

Heading

```
subroutine issetpri ( name, iprio )
```

Parameters

integer name, iprio

name integer variable containing the sequence number of the array of which the priority must be set.

iprio new priority level ($\geq$0).

Input

iprio and name must have a value.
Array ibuffr must have been filled before.

Output

Array ibuffr has been changed.

5.6.14 Subroutine isingetp

Description

Get the pointer corresponding to an integer array. It is checked whether this array is active and has priority number $\geq$ 10

Heading

`function isingetp ( name )`

Parameters

integer name, isingetp

name integer variable containing the sequence number of the array of which the starting address must be received.

isingetp starting address of array in ibuffr.

Input

name must have a value.
Array ibuffr must have been filled before.

Output

isingetp has got a value.

5.6.15 Subroutine isdpgetp

Description

Get the pointer corresponding to a double precision array. It is checked whether this array is active and has priority number $\geq$ 10

Heading

`function isdpgetp ( name )`

Parameters

integer name, isdpngetp

name integer variable containing the sequence number of the array of which the starting address must be received.

isdpgetp starting address of array in buffer.

Input

name must have a value.
Array ibuffr must have been filled before.

Output

isdpgetp has got a value.

5.6.16 Subroutine isinleng

Description

Get the length of an integer array

Heading

```
function isinleng ( name )
```

Parameters

integer name, isinleng

name integer variable containing the sequence number of the array of which the starting address must be received

isinleng length of integer array in ibuffr

Input

name must have a value.
array ibuffr must have been filled before.

Output

isinleng has got a value.

5.6.17 Subroutine isdpleng

Description

Get the length of a double precision array

Heading

```
function isdpleng ( name )
```

Parameters

integer name, isdpleng

name integer variable containing the sequence number of the array of which the starting address must be received

isinleng length of double precision array in ibuffr

Input

name must have a value.
array ibuffr must have been filled before.

Output

isdpleng has got a value.

Chapter 6

Parallel aspects

6.1 introduction

The communication between host and node must be performed in a unique and simple way independent of the parallel architecture actually used. For that reason we define our own interface subroutines. The actual subroutines underlying these interface subroutines, however, are at the moment based on PVM, since we expect that PVM is the ad-hoc standard for parallel communication at this moment. The receive subroutine must detect whether an error is received or a standard message. Errors must be printed by the host program only.

An extra option that we want to use is that if the node program runs at the same computer as the host, the node program will be treated as a subroutine of the host program. In that case we do not want to send and receive data form host to node and vice-versa but only to copy data from one to another. Since this possibility should be transparent for the programmer the send and receive subroutines should be able to detect themselves whether they are at a node or at the host.
Information about the parallel process is stored in the common cisparal 6.3.
In Section 6.2 the tools for communication are described. Section 6.3 gives a description of the common cisparal. In Section 6.4 it is described how the data to be sent from host to node and vice versa are stored in the sending buffer. If the node program runs as a subroutine of the host program, data does not have to sent but should be copied.

6.2 Description of the communication subroutines

Following the PVM structure we define the following subroutines:

istparhs 6.2.1	Start parallel application at host and activate nodes
istparnd 6.2.2	Start parallel application at node
issend 6.2.3	Send data to node or host
isreceiv 6.2.4	Receive data from node or host
isexitpm 6.2.5	Exit parallel application node or host

6.2.1 Subroutine istparhs

Description

Initialize parallel process (i.e. call pvmstart).
Start parallel application at host and activate nodes.
Initialize the common cisparal 6.3.

Heading

```
subroutine istparhs ( nnodes, namenodp )
```

Parameters

integer nnodes

character *(*) namenodp

namenodp name of node program

nnodes Number of nodes that must be activated

Input

NNODES must have a value

Output

The parallel process at the host has been started.
The host has been initialized.
The nodes have been activated.

6.2.2 Subroutine istparnd

Description

Start parallel application at node. Initialize the common cisparal.

Heading

```
subroutine istparnd
```

Output

The parallel process at the node has been initialized.
The common block cisparal has been initialized.

6.2.3 Subroutine issend

Description

Send data to node or host.

Heading

```
subroutine issend ( ifirst, itype, length, rarray, iarray, carray,
                    nodenr )
```

Parameters

integer ifirst, itype, length, iarray(*), nodenr

double precision rarray(*)

character *(*) carray(*)

ifirst With this parameter it is indicated if the data to be sent is the first in a row, somewhere in the middle or the last of a row. Possible values:

- **0** The data to be sent is the only data to be sent. The actual send is performed.
- **1** The data to be sent is the first of a series of data. More data will be sent in the same actual send.
- **2** The data to be sent is not the first nor the last of a series of data. More data will be sent in the same actual send.

3 The data to be sent is the last of a series of data. The actual send is performed.

itype Type of data to be sent. Possible values:

-3 The data is of the type output

-2 The data is a warning

-1 The data is an error message

0 The data consists of strings

3 The data consists of integers (int*4)

6 The data consists of reals (doubles)

The possibility itype < 0, is not meant for the user but only for some special subroutines dealing with output, warnings and error messages.

length Number of data to be sent, i.e. number of integers, strings or reals.
length = 0, is allowed.

nodenr sequence number of node to which message must be sent.
If the message must be sent to the host, nodenr should be 0.

iarray integer array to be sent in case itype = 3

rarray real array to be sent in case itype = 6

carray array of strings to be sent in case itype = 6

Input

ifirst, itype and length must have a value.
Depending on the value of itype one of the arrays iarray, rarray or carray must have been filled.

The parameter IBLOCKRF in common cisnas 16.26 is used to identify the block sequence number. This number is put in array INFO in common block CISCOM and is visible in the parameter list of ISRECEIV. Hence the user must give IBLOCKRF a value before the call of ISSEND.

Output

If ifirst = 0 or 1 the sending buffer has been initialized.
If ifirst = 0 or 3 the buffer has been sent.
If ifirst = 1 or 2 information has been put into the sending buffer.
The positions 1, 8, 9 and 10 of array INFO in common block CISCOM may have been changed.

Remarks

Errors, warnings and print output at the node will also activate send buffer.
In case of a warning or print output this implies adding of the message to the buffer.
In case of an error this implies that the message is sent immediately, and as a consequence data to be sent afterwards will not be sent anymore. An error at a node will be treated as a fatal error, which implies that the node program is halted after the sending of the message and the parallel process is stopped.
PVM allows for one send buffer only. As a consequence packing of arrays to be sent must be restricted to one node at a time. The actual send must be performed before packing for a new node.
The parameter msgtype of PVM has been deliberately avoided. At this moment I think that the programmer himself must identify his messages, for example by providing block numbers, face sequence numbers etcetera.

6.2.4 Subroutine isreceiv

Description

Receive data from node or host.

Heading

```
subroutine isreceiv ( ifirst, itype, length, rarray, iarray, carray,
                      inode, iblock )
```

Parameters

integer ifirst, itype, length, iarray(*), inode, iblock

double precision array(*)

character *(*) carray(*)

ifirst with this parameter it is indicated if the data to be received is the first in a row, somewhere in the middle or the last of a row.
Possible values:

1 The data to be received is the first of a series of data. The actual receive is performed.

2 The data to be received is not the first of a series of data.

3 The data to be received is the first of a series of data. Only the information array info is read and the length of the next array is returned.

4 The data to be received is not the first of a series of data. Only the information array info is read and the length of the next array is returned.

5 The data to be received is not the first of a series of data. It is supposed that the information array info has already been read by a previous call of isreceiv. Only the actual data are read.

itype Type of data to be received.
Possible values:

-3 The data is of the type output

-2 The data is a warning

-1 The data is an error message

0 The data consists of strings

3 The data consists of integers (int*4)

6 The data consists of reals (doubles)

The possibility itype < 0, is not meant for the user but only for some special subroutines dealing with output, warnings and error messages.

length number of data to be received, i.e. number of integers, strings or reals.

iarray integer array to be received in case itype = 3

rarray real array to be received in case itype = 6

carray array of strings to be received in case itype = 0

inode Node number of sending node. If 0 the message has been sent from the host.

iblock Actual block number corresponding to the data.

Input

ifirst, itype and length must have a value.

Output

If ifirst = 1 or 3 the receiving buffer has been filled.
If ifirst = 1, 2 or 5 one of the arrays iarray, rarray or carray has been filled.
If ifirst = 3 or 4 length has gotten a value.
The parameters inode and iblock have got a value.
Array info has been filled.

Remarks

The parameters itype and length are only meant for checking purposes. Actually these parameters have already been set by issend.
If an error has been sent by the node to the host, an error message is printed. We do not halt the process yet, but after receiving data and as a consequence errors from all nodes the process must be terminated by the global process. If a warning or a print output is received by the host, these are printed and the computation resumed.

6.2.5 Subroutine isexitpm

Description

Exit parallel application at node or host.

Heading

```
subroutine isexitpm
```

Output

The parallel process at the host or node has been halted.

6.3 Common block cisparal

All necessary information about the parallel process is stored in common block cisparal. Common cisparal has the following shape:

```
integer maxnodes
parameter ( maxnodes=100 )
integer nodeid(0:maxnodes), iacnodes, inodenr, idnr
common /cisparal/ nodeid, iacnodes, inodenr, idnr
```

Description of the parameters:

iacnodes Number of nodes active

idnr Identification number of present node.

inodenr Sequence number of present node. If inodenr = 0, it concerns the host.

maxnodes Maximum number of nodes

nodeid Array containing an identification number for each node
nodeid(0) refers to the host.

Extra parameters may be defined in the future. The present list is preliminary.

6.4 Organization of the send buffer in case of parallel processing

The tools described in this chapter can only work properly, if it is defined how information is transported. In this section we shall discuss a possible storage scheme for the transport. We assume that some type of buffering is used to transport the actual data. How this buffering takes place will not be discussed. At this moment we shall use PVM and its corresponding buffer mechanism. So in fact we limit ourselves to how information is stored in this buffer. Although we know exactly what information we are sending or receiving it should not be necessary to send extra information about the data to be sent. However, in practice the situation is somewhat more complex since also error messages and warnings may be transported and it is not a priori known if we should expect data or warnings respectively errors. Furthermore, it would be nice if there some mechanism to check if the data that has been send is of the correct type. For that reason it has been decided that a message to be sent consists of a number of submessages. The number of submessages is completely free. Each submessage itself consists of an array of length 10 containing information about the actual data followed by the actual data itself. Since we use a buffer mechanism this does not actually lead to a large overhead. Hence the structure of the complete message will be as follows:

administration subm 1	data subm 1	administration subm 2	data subm 2	...

The following contents of this administration array of length 10 are proposed:

1 **itype** type of message
Possible values:

- **0** The data consists of strings
- **3** The data consists of integers (int*4)
- **6** The data consists of reals (doubles)
- **-1** The message to be sent is an error message
- **-2** The message to be sent is of the type warning
- **-3** The message to be sent is of the type output

2 **length** Number of data to be sent, i.e. number of integers, strings or reals

3-7 Not yet defined (set them to 0)

8 **iblock** Block sequence number, see cisnas

9 **inodenr** Sequence number of node, see cisparal 6.3

10 **ifirst** parameter indicating if the submessage is the last message to be sent or received (1) or not (0).

In case of an error, a warning or a print command the definition is a little bit different from the definition above. In that case we store the following information in the positions 2-5:

2 mesnum, i.e. message sequence number. This is either the error number, the warning number or the format sequence number for the print subroutine.

3 nints, see subroutines iserrsub, iswarsub and isprint

4 nreal, see subroutines iserrsub, iswarsub and isprint

5 nchar, see subroutines iserrsub, iswarsub and isprint

In that case the array of length 10 is followed by at most three arrays of length nints, nreal respectively nchar containing the actual information corresponding to the error message or warning. This information is extracted from the common blocks cismessc and cismessg.

In the standard case subroutine issend contains enough parameters to fill the administration array. However, in case positions 3 to 9 are used, as is for example the case for error messages and warnings, the parameter list is too short. In order to avoid a constant updating of the parameter list it has been decided to introduce an extra common block ciscom 6.4.1 for the storage of the information array of length 10. The user may fill this array himself before calling issend, however, the positions 1 and 10 are always filled by issend. If itype > 0, also position 2 is filled by issend. issend does not reset this common ever.
The receiving subroutine isreceiv reads the information array and puts the contents also in common ciscom. For a description of ciscom see Section 6.4.1

6.4.1 Common block ciscom

Common block ciscom is only meant to store the information array used by the communication subroutines issend and isreceiv. In fact it contains a copy of this information array as described in Section 6.4. Common ciscom has the following shape:

```
integer info
common /ciscom/ info(10)
```

Array info is completely described in Section 6.4.

Chapter 7

Error messages

7.1 Introduction

In this chapter it is described how the error messages in ISNaS are treated. The error message handling is based upon SEPRAN subroutines. These subroutines are copied, adapted and provided with the suffix is in order to distinguish them from SEPRAN.
Error messages are defined by the developer of a subroutine. Error messages are not printed directly by the programmer, but are printed with the help of a subroutine. Error messages are stored in a file, and are provided by the programmer to the manager of this file in the form of an ASCII file. Error messages are treated according to strict rules as described in this chapter. The main idea of the treatment of the error messages is that each error message must be self explaining, and if possible must contain hints to repair the error. So if possible a cause or a possible cause of the error must be given. Furthermore it is assumed that error messages give as much as possible information concerning the relevant data. Besides that, it is assumed that the error message must be given immediately on the screen or in the output file. As a consequence the user does not need a manual containing information about the error messages. Besides error messages, ISNaS also uses warnings. The only difference between the two is that error messages stop the program after one or more appearances, whereas warnings do not influence the program at all. So in the remaining part of this section we shall only speak about error messages.

In order to fulfil these requirements all error messages are put into one error message file. The exact definition of the file is described in Section 7.2. Besides that, all error messages should be available in a standard ASCII file. This ASCII file is read by the main program ismakef4 and its contents is put into the direct access error message file. See 7.3 for a description of how to use ismakef4. Section 7.3 also contains a description of the ASCII file. Besides that, 7.3 contains the procedure to extend (in fact overwrite) the error message file. To identify the various errors and warnings each error and/or warning is provided a unique sequence number.

In a parallel environment the situation is somewhat more complicated. In that case an error at the host should be treated in the standard way, however, an error at a node program can not be printed, since a node does not perform actual output. For that reason the error message is send to the host, and the node program is stopped. The host program prints the error message provided by extra information about the node and the block where the error message has been found.

Another problem related to error messages is the amount of information that is necessary in order to interpret the error message. In general it is not sufficient to tell what error has occurred but also where and when. For that reason the error message will be provided by a trace back containing a list of calling subroutines as well as some extra information telling on which node (or host) the error occurred, in which block of the multiblock process at what actual time and possibly at which iteration in an iteration process. For some subroutines extra information must be provided by the routine itself. For example if the solver crashes because the matrix is singular, it is necessary to know which matrix it concerns. The solver does not know anything about the

matrix, but the calling subroutine or program does. As a consequence the solver subroutine should leave with a return code indicating the type of error and the calling subroutine should actually call the error subroutine.
In order to avoid double use of error messages it is necessary that some agreement about the error message numbers is made. A provisional agreement is described in 7.4.
For the actual printing of error messages in a subroutine, the subroutines iserrsub 7.5.1 (errors) and iswarsub 7.5.2 (warnings) must be used, as described in 7.5.
Finally in Section 7.6 some information about how to deal with parallel processing is given.

7.2 Definition of the error message file.

The error message file is a direct access file, containing unformatted data. The reference number for the file is stored in IREFER in common block cisnas. For the error message file a fixed record length of 80 bytes is used.
The error message file is defined as follows:

The first record contains one number: the largest error number (maxerr) that is stored in this file.

The next (maxerr+nrec4-1)/nrec4 records contain the starting record numbers of the error messages on the direct access file consecutively. In fact these records may be considered as an integer array istart of length maxerr+1. nrec4 defines the number of integers that can be stored in one record of the error message file. See common cmachn. Array istart is filled as follows:

The number of records needed for the error message "i" is equal to istart(i+1)-istart(i). If istart(i+1) - istart(i) = 0, this means that the error message is not available. Hence gaps in the error messages are allowed.

istart(i) gives the starting record number of the error message "i".

Finally the last records contain the error messages stored sequentially.

The $, & and % fields as defined in 7.3 are also stored in the error message file. They are replaced by integers, reals and characters by subroutine iserrsub or iswarsub as described in 7.5.

7.3 Creation and updating of the error message file.

In order to create the error message file it is necessary to define the error messages and to put them in an ASCII file. Program ismakef4 may be used for creating and overwriting this error message file. In fact the error message file is never updated. A new version overwrites the old version of the error message file. The contents of the error message file is described in 7.2. The ASCII file containing all error messages in a readable form must have the following contents:
For each error:

Error number: The error number must start in column 1

Followed by (Next line):
Text describing the error message
The text must start in column ≥ 2 and may not exceed column 79. The number of lines to be used for the error message is arbitrary.

Error numbers must be given in the natural sequence, however, gaps between error numbers are permitted.

In the ASCII file the characters $, & and % have a special meaning. The error and warning subroutines replace each set of subsequent $-signs, by the integers provided in the array ints which

is input of these subroutines. The i^{th} set of $-signs in an error record is replaced by the i^{th} entry in array ints. The number of positions used by the integer is equal to the number of subsequent $-signs. In the same way &-signs are replaced by reals (array reals) and %-signs are replaced by characters (array chars).
With respect to the $, & and % signs the following rules are applied.

If only one $, & and % sign is given the integer, real or string is substituted instead of this sign, using as much positions as needed. As a consequence the layout of the error message may be different from the one in the ASCII file. In fact the lines in the error message are extended until 80 characters are reached. In order to force a new line in the output message use the @ symbol, which is interpreted as a hard return. If more than one $, & or % signs are used behind each other, for example $$$, then the actual value is placed in exactly these number of positions. In this specific example this means that exactly three integer positions are used. In the case of a & sign in general an e-format is used to print the number. In order to force a f-format it is necessary to put a decimal point between the & signs. The number of & signs behind the decimal point defines the number of digits behind the decimal point. Hence &&&.&& produces an f-format with three digits before and two behind the decimal point. See also subroutines iserrsub and iswarsub (7.5).

Program ismakef4 reads the ASCII file and creates or overwrites the direct access error message file. All error messages must be provided on one ASCII file. For a description of the main program ismakef4, refer to the appendix.
For a good management of the error message file it is necessary to follow strict rules for updating the file. At this moment we make the following arrangements:

There is only one manager for the error message file. All new error messages are provided in separate files. These new error message numbers must be unique. The manager adds the new error messages to the ASCII error messages file and overwrites the direct access file by program ismakef4. In the future running ismakef4 might be part of an automatic installation or update procedure.

Example of an ASCII error message file:

```
1:
 ichois has wrong value ($)
2:
 length of array too small
3:
 Element has $ nodal points, whereas for this type number $ nodes are
 required.  Element number: $, type number $, element group $
4:
 nelgrp<1 or nelgrp>99
5:
 Number of degrees of freedom in a point is $, which is less than 1 or
 more than 15. Element group $
6:
 no nodal points are created along the curve. $ th curve
7:
 Number of arrays of special structure is $, which is less than 1 or
 more than 10. Element group $
8:
 iplot has value $, which is less than zero
```

Warning:

The present version of the program ismakef4 does not contain any precaution against empty messages, i.e error message number, immediately followed by a new error message number. So one must be careful to avoid this error in the creation of the ASCII file. No error message is produced by ismakef4 for this possibility.

7.4 Agreements with respect to the filling of the error message file.

As described earlier (see 7.2 and 7.3) it is necessary that all error messages have unique numbers. In order to ensure such a unique numbering it is necessary to make fixed agreements with respect to the use of error message numbers for various applications.
At this moment the following subdivision has been agreed:

```
1    -  499:  General errors in administration
500  -  999:  General errors in input
1000 - 1099:  Errors in linear solvers of Krylov type
1100 - 1199:  Errors in linear solvers of Multigrid type
1200 - 1299:  Errors in turbulence modelling of k-eps type
1300 - 1399:  Errors in turbulence modelling of large eddy type
1400 - 1499:  Errors in memory management subroutines
1500 - 1599:  Errors in parallellization of subdomain solver
1600 - 1699:  Errors in domain decomposition
1700 - 1799:  Errors in parallel tools
```

7.5 Available subroutines with respect to error messages and warnings

At this moment ISNaS contains two subroutines that may be utilized for printing errors and warnings. Error messages are printed by iserrsub and warnings by iswarsub.
Since an important part of the information is given in the commons cismessc and cismessg, also subroutines are available to fill these commons. In fact there is no need to store the commons cismessc 7.5.3 and cismessg 7.5.3 in your subroutines. The subroutines iserrint 7.5.4, iserreal 7.5.5 and iserrchr 7.5.6 store integers, reals and strings respectively into the common blocks. To create the name of the calling subroutine in a tree structure the subroutines iseropen 7.5.7 and iserclos 7.5.8 must be used.

A typical ISNaS subroutine has the following shape:

```
      subroutine .......

          .
          .
          .        Standard declarations etcetera according to the document
          .        "Programming requirements"
          .

      call iseropen ( 'name of subroutine' )

          .
          .
```

```
      .    statements
      .
      .

Error messages in the following sense:

  call iserrint ( ..., 1 )
  call iserrint ( ..., 2 )
      .
      .
  call iserrchr ( ... , ... )
  call iserrsub ( num, nint, nreal, nchar )
      .
      .
      .

  call iserclos ( 'name of subroutine' )
```

The subroutines iseropen and iserclos must be called exactly once in each subroutine.
For main ISNaS subroutines, in which also the time is printed and the priorities are decreased, iserclos may be replaced by iserclmn.
The subroutines iserrsub and iswarsub may be used as described in the next sections.

7.5.1 Subroutine iserrsub

Description

iserrsub prints an error message.
iserrsub does not actually stop the program except when more than maxerr errors occur.
The programmer may use common block cisnas to find out whether an error occurred. He himself is responsible for the stopping of the program based on this information. Stopping of the program should be performed by subroutine isstop as described in the appendix, in order to get make sure that all ISNaS files are properly closed.

Heading

`subroutine iserrsub ( mesnum, nints, nreal, nchar )`

Parameters

integer mesnum, nints, nreal, nchar

mesnum Error number.

nchar Number of character strings.

nints Number of integers.

nreal Number of reals.

Input

mesnum, nints, nreal and nchar must have a value.
The corresponding reals, integers and characters must be stored in the common blocks cismessc and cismessg. These common blocks may be filled directly, but it is preferred to use the subroutines iserrint, iserreal and iserrchr.

Output

Print of the error message.

7.5.2 Subroutine iswarsub

Description

iswarsub prints a warning.
The printing of warnings by subroutine iswarsub is automatically suppressed after maxwar warnings. The programmer may use common block cisnas to find out whether warnings have been printed and how many. iswarsub uses the same error message file as iserrsub. So an error in the file printed by iserrsub, may turn into a warning if printed by iswarsub.

Heading

```
subroutine iswarsub ( mesnum, nints, nreal, nchar )
```

Parameters

integer mesnum, nints, nreal, nchar

mesnum Warning number.

nchar Number of character strings.

nints Number of integers.

nreal Number of reals.

Input

mesnum, nints, nreal and nchar must have a value.
The corresponding reals, integers and characters must be stored in the common blocks cismessc and cismessg. These common blocks may be filled directly, but it is preferred to use the subroutines iserrint, iserreal and iserrchr.

Output

Print of the warning.

7.5.3 The common blocks cismessc and cismessg

The common blocks cismessc and cismessg are defined as follows:

```
      character * 80 chars
      character * 8 namsubs
      common /cismessc/ chars(1000), namsubs(20)
      save /cismessc/
c
c                  /cismessc/
c     Contains characters for the error message subroutines
c     Must be used in co-operation with common block cismessg
c
c     chars     Array containing character information for error messages
c               At most 10 positions are available
c     namsubs   In this array the trace back of the subroutines is stored.
c               At level 1 the top of the calling tree is stored at level
c               levelsub (see cismessg) the actual subroutine (i.e. the bottom
c               of the tree)
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      integer ints, levelsub
```

```
      double precision reals
      common /cismessg/ reals(1000), ints(1000), levelsub
      save /cismessg/
c
c                 /cismessg/
c     Contains extra information for the error message subroutines
c     Must be used in co-operation with common block cismessc
c
c     ints      Array containing integer information for error messages
c     levelsub  Level of present subroutine for trace back
c     reals     Array containing real information for error messages
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Although these commons may be filled by hand, it is preferred to use the subroutines iseropen, iserclos, iserrint, iserreal and iserrchr to fill these commons.

7.5.4 Subroutine iserrint

Description

iserrint is an auxiliary subroutine to iserrsub. It must be called prior to iserrsub and sets the message variable int. errvar is the number of the variable to be set (errvar $\leq$ 10).

Heading

```
subroutine iserrint ( int, errvar )
```

Parameters

integer int, errvar

int Integer variable to be put into array ints

errvar Sequence number of variable in ints

Input

int and errvar must have a value

Output

The value of the integer is stored in common cismessc

7.5.5 Subroutine iserreal

Description

iserreal is an auxiliary subroutine to iserrsub. It must be called prior to iserrsub and sets the message variable eal. errvar is the number of the variable to be set (errvar $\leq$ 10).

Heading

```
subroutine iserreal ( eal, errvar )
```

Parameters

integer errvar

double precision eal

eal Double precision variable to be put into array reals

errvar Sequence number of variable in reals

Input

eal and errvar must have a value

Output

The value of the real is stored in common cismessc

7.5.6 Subroutine iserrchr

Description

iserrchr is an auxiliary subroutine to iserrsub. It must be called prior to iserrsub and sets the message variable chr. errvar is the number of the variable to be set (errvar $\leq$ 10).

Heading

```
subroutine iserrchr ( chr, errvar )
```

Parameters

character * (*) chr

integer errvar

chr Character variable to be put into array chars

errvar Sequence number of variable in chars

Input

chr and errvar must have a value

Output

The value of the string is stored in common cismessg.

7.5.7 Subroutine iseropen

Description

ISNaS subroutine, that adds the name of the local subroutine to the list of subroutines stored in namsubs. The parameter levelsub is raised by one.

A call of iseropen must always be followed by a call to iserclos or iserclmn in the same subroutine!

Heading

```
subroutine iseropen ( namelc )
```

Parameters

character *(*) namelc

namelc Name of local subroutine that must be concatenated

Input

namelc must have a value.

Output

The name of the subroutine has been stored in namsubs.
The parameter levelsub has been raised.

7.5.8 Subroutine iserclos

Description

ISNaS subroutine, that removes the name of the local subroutine from the parameter name in common cismessg/cismessc by reducing levelsub

A call of iserclos must always be preceded by a call to iseropen in the same subroutine, and with the same name!

Heading

```
subroutine iserclos ( namelc )
```

Parameters

character *(*) namelc

namelc Name of local subroutine that must be removed

Input

namelc must have a value.

Output

The parameter levelsub has been decreased.

7.5.9 Subroutine iserclmn

Description

subroutine iserclmn does the final closing of a main ISNaS subroutine. It is supposed that the subroutine has been opened with iseropen and that iserclos has not been called iserclmn performs the following tasks:
call the timing subroutine depending on the parameter itime.
decrease all priorities of the arrays in ibuffr 5.5.
call iserclos, i.e. release the name of the main subroutine from name.

Heading

```
subroutine iserclmn( namenw )
```

Parameters

character *(*) namenw

namenw Name of calling subroutine

Input

namelc must have a value

Output

The parameter levelsub has been decreased.

7.6 Special arrangements with respect to parallel processing

Since the ISNaS program is developed such that the node programs may run on other computers than the host program it is not possible to print the error message unconditionally. If the node program is on the same computer the error message or warning is printed, however, if the node program runs on a different computer information of the message is transported by PVM to the host program, which expands the message. This means that the error message subroutines must know at which computer they are called and what actions should be taken. To the programmer this must be transparent. How the transports for parallel computers are arranged is described in Chapter 6.

Chapter 8

Timing subroutines

8.1 Introduction

Timing in a parallel environment is a tricky matter. The question is of course in which timing are we interested and how do we receive and store this information. Since in parallel computing both cpu time and wall clock time (communication time) are of interest it is best to store the information of both parts.
At this moment, however, we do not store nor compute the wall clock time. In fact the only time that is computed is the CPU time per subroutine and per processor.

Information about the CPU time used is stored in common block cistime 8.3.
This common is initialized in the starting subroutine and updated in the main subroutines. It contains an array `time` of length 10 in which the initial CPU time for a subroutine may be stored. The procedure is the following:
at the start of a main subroutine the CPU time is stored in `time(1)`
at the end of the main subroutine the CPU time minus the starting CPU is printed.
If one wants to consider the computation time of sub parts of the main subroutine, other positions in array `time` must be used. With the aid of a sequence number these positions may be indicated.
In 8.2 the available timing subroutines are treated. Section 8.3 describes the common block cistime.

8.2 Description of the timing subroutines

The following timing subroutines are available:

isiniclk 8.2.1	Initialize the timer and array timeisns
issetclk 8.2.2	Set the starting time for a main subroutine
isaskclk 8.2.3	Compute and print the time for a main subroutine

isiniclk should be called by the starting subroutine (ishstart and isnstart) only.
issetclk must be called at the beginning of a main subroutine and isaskclk at the end of a main subroutine.

8.2.1 Subroutine isiniclk

Description

set timer equal to zero.
Initialize array time in common cistime.

Heading

```
subroutine isiniclk
```

Output

Array time has been initialized.

8.2.2 Subroutine issetclk

Description

set timer parameters for specific subroutine equal to zero

Heading

```
subroutine issetclk ( iseqnr )
```

Parameters

integer iseqnr

iseqnr Sequence number with respect to common cistime.

Input

iseqnr must have a value The call of issetclk must be preceded by a call to isiniclk.

Output

The actual CPU time at the call of issetclk is stored in time(iseqnr)

8.2.3 Subroutine isaskclk

Description

Compute and print the CPU time used by the main subroutine.

Heading

```
subroutine isaskclk ( iseqnr, text )
```

Parameters

integer iseqnr

character $*(*)$ text

iseqnr Sequence number with respect to common cistime.

text Text that is printed before printing the CPU time. Actually exactly 39 positions of text are printed in order to get a fixed format.

Input

iseqnr and text must have a value. The value must of iseqnr must be exactly the same as that used in the call of issetclk in the same subroutine.

Output

Print of the CPU time used.

8.3 Common block cistime

All necessary information about the timing is stored in common block cistime. Common cistime has the following shape:

```
integer maxsub
parameter ( maxsub=10 )
double precision time(maxsub)
common /cistime/  time
```

Array time(iseqnr) is filled by the CPU time at the call of subroutine issetclk with parameter iseqnr.

Chapter 9

Print subroutines

9.1 Introduction

One of the problems we are faced with is that printing at a node is not longer possible in a parallel environment. In fact, just as is the case with the error subroutines it is necessary to code the print information, send it from node to host and then actually print it. For the programmer this process is hided, which means that there is a print subroutine which decides itself whether it is at the host or at a node. In the last case the printing subroutine sends information to the host using the send mechanism. The receive mechanism recognizes the message sent and carries out the actual print without interrupting the process actually performed.

In this chapter we describe this general printing subroutine isprint 9.2.1 as well as its underlying subroutines isgenpr 9.2.2 and isuserpr 9.2.3. These subroutines perform the actual printing. isgenpr is used to print general ISNaS data according to fixed formats. If extra general prints are necessary this is the subroutine to be extended. isuserpr is subroutine that must be provided by the user. This subroutine is meant for special (local) output only.
Besides these general printing subroutines, ISNaS has a number of special printing subroutines which are generally meant for debugging purposes. In fact these subroutines print one or two of the standard ISNaS arrays. They may be placed anywhere in the code.

The following special subroutines are available:

isprar01 9.3.1	Low level print routine for a two-dimensional field
isprar02 9.3.2	Low level print routine for a three-dimensional field
isprar03 9.3.3	Low level print routine for a matrix in R^2
isprar04 9.3.4	Low level print routine for a matrix in R^3
isprimt1 9.3.5	Print a matrix in R^2 including the heading
isprimt2 9.3.6	Print a matrix in R^3 including the heading
isprirh1 9.3.7	Print a solution vector in R^2 including the heading
isprirh2 9.3.8	Print a solution vector in R^3 including the heading

9.2 General print subroutines

9.2.1 Subroutine isprint

Description

General print subroutine. Prints reals integers and strings in a general format or in the format provided by the user.

Heading

```
subroutine isprint ( mesnum, nints, intarr, nreal, realarr, nchar, chararr )
```

Parameters

integer mesnum, nints, nreal, nchar, intarr(nints)

double precision realarr(nreal)

character *(*) chararr(nchar)

chararr Array of character strings containing the strings that must be substituted in the print format.

intarr Array of integers containing the integers that must be substituted in the print format.

mesnum Sequence number of the type of format.
If $1 \leq mesnum \leq 100$ isprint calls the user print subroutine isuserpr, in which the user may define his own output formats.
For $mesnum \geq 100$ the standard ISNaS print subroutine isgenpr is called, which prints according to predefined formats. The relation between mesnum and predefined format is given in isgenpr.

nchar Number of character strings stored in array chararr.

nints Number of integers stored in array intarr.

nreal Number of reals stored in array realarr.

realarr Array of reals containing the reals that must be substituted in the print format.

Input

mesnum, nints, nreal and nchar must have a value.
Array intarr positions 1 to nints must have been filled.
Array realarr positions 1 to nreal must have been filled.
Array chararr positions 1 to nchar must have been filled.

Output

Print of all information in the format required at the host.

9.2.2 Subroutine isgenpr

Description

Actual print subroutine. Prints reals integers and strings in a special format depending on the value of the parameter mesnum.

Heading

```
subroutine isgenpr ( mesnum, nints, intarr, nreal, realarr, nchar, chararr )
```

Parameters

integer mesnum, nints, nreal, nchar, intarr(nints)

double precision realarr(nreal)

character *(*) chararr(nchar)

chararr Array of character strings containing the strings that must be substituted in the print format.

intarr Array of integers containing the integers that must be substituted in the print format.

mesnum Sequence number of the type of format.
isgenpr is for values of mesnum > 100. For each value a specific format is defined.
The following values of mesnum > 100 are available:

101 print arrays iinput and rinput

102 print array coor

103 print array solut

104 print array geom

105 print array matrix

106 print array rhside

107 print array coefs

108 print information from the linear solver

109 print residuals from the linear solver

200 print CPU time

nchar Number of character strings stored in array chararr.

nints Number of integers stored in array intarr.

nreal Number of reals stored in array realarr.

realarr Array of reals containing the reals that must be substituted in the print format.

Input

mesnum, nints, nreal and nchar must have a value.
Array intarr positions 1 to nints must have been filled.
Array realarr positions 1 to nreal must have been filled.
Array chararr positions 1 to nchar must have been filled.

Output

Print of all information in the format required at the host.

9.2.3 Subroutine isuserpr

Description

User print subroutine. Prints reals integers and strings in a format provided by the user.
The user must program subroutine isuserpr himself.

Heading

```
subroutine isuserpr ( mesnum, nints, intarr, nreal, realarr, nchar, chararr )
```

Parameters

integer mesnum, nints, nreal, nchar, intarr(nints)

double precision realarr(nreal)

character *(*) chararr(nchar)

chararr Array of character strings containing the strings that must be substituted in the print format.

intarr Array of integers containing the integers that must be substituted in the print format.

mesnum Sequence number of the type of format.
This subroutine is only called if $1 \leq mesnum \leq 100$.

nchar Number of character strings stored in array chararr.

nints Number of integers stored in array intarr.

nreal Number of reals stored in array realarr.

realarr Array of reals containing the reals that must be substituted in the print format.

Input

mesnum, nints, nreal and nchar must have a value.
Array intarr positions 1 to nints must have been filled.
Array realarr positions 1 to nreal must have been filled.
Array chararr positions 1 to nchar must have been filled.

Output

Print of all information in the format required at the host.

9.3 Special print subroutines

9.3.1 Subroutine isprar01

Description

This subroutine prints the contents of a double precision vector defined on a two-dimensional rectangular grid, possibly provided with virtual positions.
This subroutine is especially meant as under routine for other printing subroutines.

Heading

```
subroutine isprar01 ( array, nvirtual, ni, nj, ndim )
```

Parameters

integer nvirtual, ni, nj, ndim

double precision array(1-nvirtual:ni+1+nvirtual, 1-nvirtual:nj+1+nvirtual,1:ndim)

character $*(*)$ text

array Actual array to be printed. The size of this array is defined by the integer parameters.

nvirtual Number of rows of virtual cells at each side of the grid.

ni Number of cells in the "i"-direction.

nj Number of cells in the "j"-direction.

ndim Number of quantities stored per point.

Input

Array array must have been filled.
The parameters nvirtual, ni, nj and ndim must have a value.

Output

Print of the array in a suitable format.

9.3.2 Subroutine isprar02

Description

This subroutine prints the contents of a double precision vector defined on a three-dimensional rectangular grid, possibly provided with virtual positions.
This subroutine is especially meant as under routine for other printing subroutines.

Heading

```
subroutine isprar02 ( array, nvirtual, ni, nj, nk, ndim )
```

Parameters

integer nvirtual, ni, nj, nk, ndim

double precision array(1-nvirtual:ni+1+nvirtual, 1-nvirtual:nj+1+nvirtual,1-nvirtual:nk+1+nvirtual,1:ndim

character $*(*)$ text

array Actual array to be printed. The size of this array is defined by the integer parameters.

nvirtual Number of rows of virtual cells at each side of the grid.

ni Number of cells in the "i"-direction.

nj Number of cells in the "j"-direction.

nk Number of cells in the "k"-direction.

ndim Number of quantities stored per point.

Input

Array array must have been filled.
The parameters nvirtual, ni, nj, nk and ndim must have a value.

Output

Print of the array in a suitable format.

9.3.3 Subroutine isprar03

Description

This subroutine prints the contents of a double precision matrix defined on a two-dimensional rectangular grid, not provided with virtual positions.
This subroutine is especially meant as under routine for other printing subroutines.

Heading

```
subroutine isprar03 ( matrix, ni, nj, nconct, ndim, itypmat )
```

Parameters

integer nvirtual, ni, nj, nconct, ndim, itypmat

double precision matrix(1:ni+1,1:nj+1,1:ndim,1:nconct)

matrix Actual matrix to be printed. The size of this array is defined by the integer parameters.

ni Number of cells in the "i"-direction.

nj Number of cells in the "j"-direction.

nconct Number of connections in a row of the matrix

ndim Number of quantities stored per point.

itypmat Type of matrix. Possible values:

1 Momentum matrix

2 Pressure of transport matrix

3 Gradient matrix

Input

Array matrix must have been filled.
The parameters ni, nj, nconct, ndim and itypmat must have a value.

Output

Print of the matrix in a suitable format.

9.3.4 Subroutine isprar04

Description

This subroutine prints the contents of a double precision matrix defined on a two-dimensional rectangular grid, not provided with virtual positions.
This subroutine is especially meant as under routine for other printing subroutines.

Heading

```
subroutine isprar04 ( matrix, ni, nj, nk, nconct, ndim, itypmat )
```

Parameters

integer nvirtual, ni, nj, nconct, ndim, itypmat

double precision matrix(1:ni+1,1:nj+1,1:nk+1,1:ndim,1:nconct)

matrix Actual matrix to be printed. The size of this array is defined by the integer parameters.

ni Number of cells in the "i"-direction.

nj Number of cells in the "j"-direction.

nk Number of cells in the "k"-direction.

nconct Number of connections in a row of the matrix

ndim Number of quantities stored per point.

itypmat Type of matrix. Possible values:

1 Momentum matrix

2 Pressure of transport matrix

3 Gradient matrix

Input

Array matrix must have been filled.
The parameters ni, nj, nconct, ndim and itypmat must have a value.

Output

Print of the matrix in a suitable format.

9.3.5 Subroutine isprimt1

Description

This subroutine prints the contents of the array matrix on a two-dimensional grid.
The heading of the output is given by text.
This subroutine is meant for debug purposes. It is assumed that the matrix and the dimension parameters are available.

Heading

```
subroutine isprimt1 ( text, matrix, ni, nj, nconct, itypmat )
```

Parameters

integer ni, nj, nconct, ndim, itypmat

double precision matrix(1:ni+1,1:nj+1,1:ndim,1:nconct)

character $*(*)$ text

text String in which the text for the heading is stored.

matrix Actual matrix to be printed. The size of this array is defined by the integer parameters.

ni Number of cells in the "i"-direction.

nj Number of cells in the "j"-direction.

nconct Number of connections in a row of the matrix

ndim Number of quantities stored per point.

itypmat Type of matrix. Possible values:

1 Momentum matrix
2 Pressure of transport matrix
3 Gradient matrix

Input

Array matrix must have been filled.
The parameters ni, nj, nconct, ndim and itypmat must have a value.

Output

Print of the matrix in a suitable format.

9.3.6 Subroutine isprimt2

Description

This subroutine prints the contents of the array matrix on a three-dimensional grid.
The heading of the output is given by text.
This subroutine is meant for debug purposes. It is assumed that the matrix and the dimension parameters are available.

Heading

```
subroutine isprimt2 ( text, matrix, ni, nj, nk, nconct, itypmat )
```

Parameters

integer ni, nj, nk, nconct, ndim, itypmat

double precision matrix(1:ni+1,1:nj+1,1:nk+1,1:ndim,1:nconct)

character ∗(∗) text

text String in which the text for the heading is stored.

matrix Actual matrix to be printed. The size of this array is defined by the integer parameters.

ni Number of cells in the "i"-direction.

nj Number of cells in the "j"-direction.

nk Number of cells in the "k"-direction.

nconct Number of connections in a row of the matrix

ndim Number of quantities stored per point.

itypmat Type of matrix. Possible values:

1 Momentum matrix
2 Pressure of transport matrix
3 Gradient matrix

Input

Array matrix must have been filled.
The parameters ni, nj, nk, nconct, ndim and itypmat must have a value.

Output

Print of the matrix in a suitable format.

9.3.7 Subroutine isprirh1

Description

This subroutine prints the contents of the array rhside on a three-dimensional grid.
The heading of the output is given by text.
This subroutine is meant for debug purposes. It is assumed that the right-hand-side vector and the dimension parameters are available.
The vector to be printed must have the structure of a right-hand-side vector, but is may also be another vector like for example a solution vector.

Heading

```
subroutine isprirh1 ( text, rhside, ni, nj,, ndegfd )
```

Parameters

integer ni, nj, ndegfd

double precision rhside(1:ni+1,1:nj+1,1:ndegfd)

character ∗(∗) text

text String in which the text for the heading is stored.

vector Actual vector to be printed. The size of this array is defined by the integer parameters.

ni Number of cells in the "i"-direction.

nj Number of cells in the "j"-direction.

ndegfd Number of degrees of freedom in the vector per "point"

Input

Array rhside must have been filled.
The parameters ni, nj and ndegfd must have a value.

Output

Print of the vector in a suitable format.

9.3.8 Subroutine isprirh2

Description

This subroutine prints the contents of the array rhside on a three-dimensional grid.
The heading of the output is given by text.
This subroutine is meant for debug purposes. It is assumed that the right-hand-side vector and the dimension parameters are available.
The vector to be printed must have the structure of a right-hand-side vector, but is may also be another vector like for example a solution vector.

Heading

```
subroutine isprirh2 ( text, rhside, ni, nj, nk, ndegfd )
```

Parameters

integer ni, nj, nk, ndegfd

double precision rhside(1:ni+1,1:nj+1,1:nk+1,1:ndegfd)

character $*(*)$ text

text String in which the text for the heading is stored.

vector Actual vector to be printed. The size of this array is defined by the integer parameters.

ni Number of cells in the "i"-direction.

nj Number of cells in the "j"-direction.

nk Number of cells in the "k"-direction.

ndegfd Number of degrees of freedom in the vector per "point"

Input

Array rhside must have been filled.
The parameters ni, nj, nk and ndegfd must have a value.

Output

Print of the vector in a suitable format.

Chapter 10

Linear solvers

Chapter 11

Building of matrices and right-hand sides

11.1 Introduction

In this section we describe the structure of the matrix construction in the ISNaS incompressible program.
With respect to the building of the matrix we distinguish between the interior cells and the so-called boundary cells, which have at least one side common to the boundary of the region. In the construction of the matrix we start by evaluating the coefficients in the matrix. Once all coefficients for an unknown are computed, these coefficients are distributed to matrix components.
In order to get a minimal execution time, it is necessary that loops over the grid cells are positioned in the lowest level subroutines. Mathematical details can be found in the ISNaS Mathematical Manual [2]

11.2 Storage and Administration

In the ISNaS incompressible code the physical domain is transformed to a rectangle. On this rectangle a uniform computational grid is defined. Figure 11.2.1 gives a typical example of the mapping from physical (i.e. curvilinear) to computational grid in two dimensions. All computa-

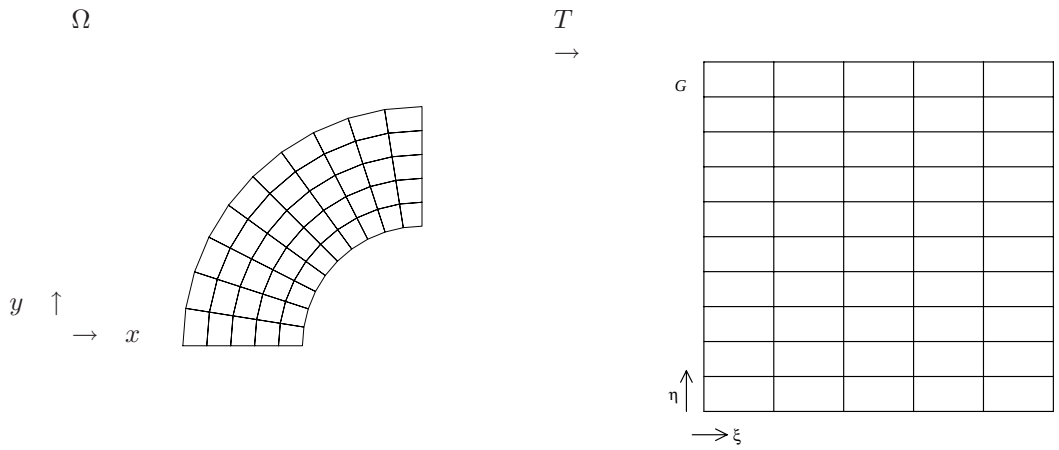

Figure 11.2.1: Boundary fitted co-ordinates and computational grid

tions are performed in the computational grid and hence the differential equations are transformed from physical grid to computational grid. The resulting solution is transformed backwards.

For the discretization of the incompressible Navier-Stokes equations on this computational grid a staggered arrangement is used. Figure 11.2.3 shows a two-dimensional example.

Numerical integration over the control volumes results in values in the integration points expressed in terms of the unknowns and in terms of derivatives of the unknowns. Take the Navier-Stokes momentum equations in tensor form:

$$T^{\alpha\beta}_{,\beta} = f^{\alpha} \tag{11.1}$$

Finite volume approximation in 2D in the computational domain, using one point numerical integration and taking the sides of the volume equal to 1 gives

$$\begin{aligned} \sqrt{g}\, f^{\alpha} &= \sqrt{g}\, \{ {\alpha \atop \beta\gamma} \} \sigma^{\beta\gamma}_{(0,0)} + (\sqrt{g}\, \sigma^{\alpha 1})_{(1,0)} - (\sqrt{g}\, \sigma^{\alpha 1})_{(-1,0)} \\ &+ (\sqrt{g}\, \sigma^{\alpha 2})_{(0,1)} - (\sqrt{g}\, \sigma^{\alpha 2})_{(0,-1)} \end{aligned} \tag{11.2}$$

See the ISNaS Mathematical Manual [2] for an explanation.
From this equation it is clear that the representation of all tensor components are needed in the centre of the volume (point (0,0)), for all types of control volumes (staggered grid). In the midpoints of the sides of the volume (points (1,0), (-1,0), (0,1) and (0,-1) see Figure 11.2.2), the representation of one typical component of the tensor is needed, also for all types of control volumes. In case of the Navier-Stokes equations these tensor components are expressed in terms

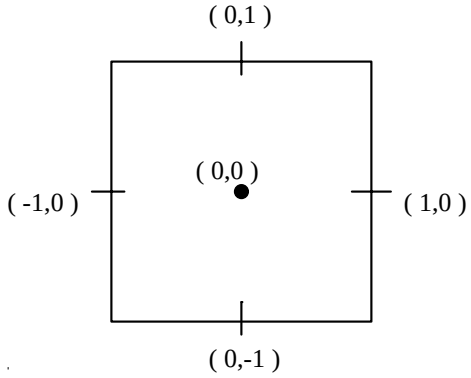

Figure 11.2.2: Integration control volume in 2D

of the unknowns and in terms of derivatives of the unknowns.

Distribution of these values to points in the computational grid where the unknowns are actual available gives the stencil which corresponds to one row of the matrix.
An example of a stencil that corresponds to the V^1-momentum control volume is given in Figure 11.2.4.
The place of the unknowns in the grid, the place of the integration points, the matrix storage and the interpolation and finite difference formulae need to be administrated.

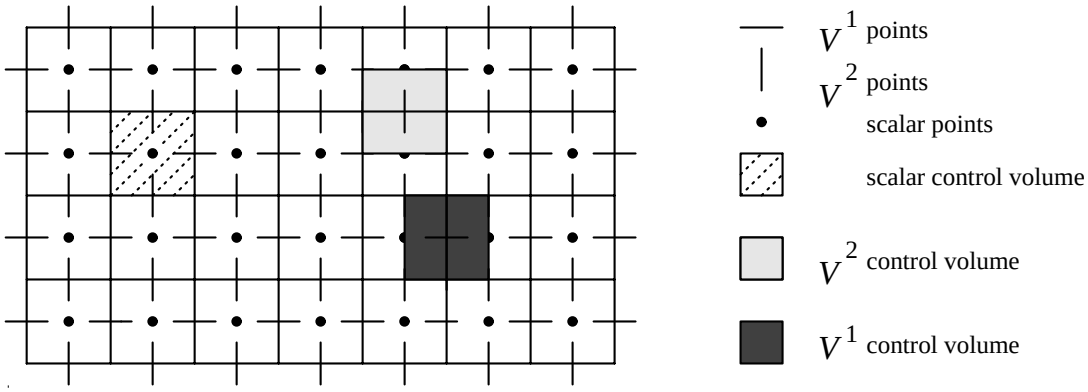

Figure 11.2.3: Arrangement of the unknowns for a staggered grid

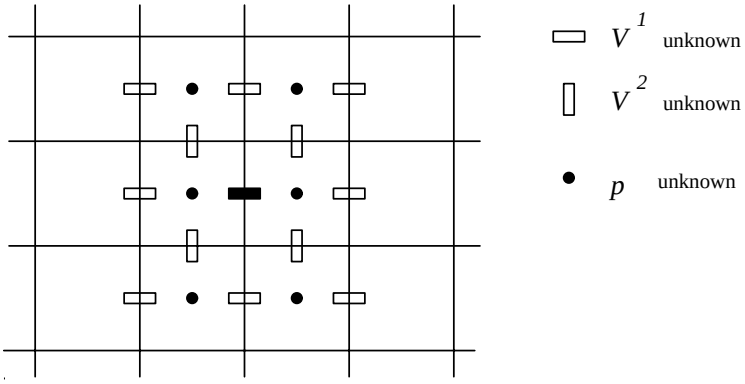

Figure 11.2.4: Stencil for V^1-momentum equation

11.2.1 Evaluation of the inner cells

In this section we consider the evaluation of the matrix and right-hand side elements for the inner cells. In order to make the program as simple as possible we treat all cells of which the centroid is part of the grid as inner cells. Effectively this means that such an inner cell may be partly outside the region. This is possible since we use at least one row of virtual cells surrounding the actual region.
Boundary conditions are applied after the matrix due to inner cells has been constructed and as a consequence it is possible that rows in the matrix created for the so-called inner cells are replaced by zeros later on.

The idea of the building of the matrix and right-hand side is as follows:
Each cell corresponds to exactly one row in the matrix or right-hand side. For each of these cells we need to evaluate 5 (in 2D) or 7 (in 3D) coefficients in integration points. For example in Equation 11.2 these are the integration points (0,0), (1,0), (-1,0), (0,1) and (0,-1). In order to construct a row of the matrix we may loop over all cells in a particular direction and then evaluate all coefficients in the integration points. However, it appears to be more efficient to loop over the integration points and within such a loop to loop over the cells. So only one coefficient at a time is computed and the contribution of these coefficients must be added to the matrix and right-hand side.
The next problem is that we may evaluate the coefficients in the integration points but that, in general, these coefficients can not immediately be placed into the matrix. Consider for example the discretized convective terms corresponding to the momentum equations. We assume that a central scheme in combination with Newton linearization is applied. According to the mathematical

manual the discretization of the convective term in the V^1 direction is given by:

$$\frac{\rho}{\sqrt{g}}(V^1)^2|_{(-1,0)}^{(1,0)} + \frac{\rho}{\sqrt{g}}V^1V^2|_{(0,-1)}^{(0,1)} + \frac{\rho}{\sqrt{g}}\{\begin{smallmatrix}1\\ \gamma\beta\end{smallmatrix}\}V^\gamma V^\beta|_{(0,0)} \tag{11.3}$$

The Newton linearization amounts to

$$V^\alpha V^\beta \approx V^\alpha \bar{V}^\beta + \bar{V}^\alpha V^\beta - \bar{V}^\alpha \bar{V}^\beta \tag{11.4}$$

where V^α is taken at the new time level and $\bar{V}^\alpha$ at the preceding one.

So we see that for example in the integration point (0,1) we have a contribution to the matrix for both velocity components. The V^1 contribution is:

$$\frac{\rho}{\sqrt{g}}\bar{V}^2 \tag{11.5}$$

and the V^2 contribution is:

$$\frac{\rho}{\sqrt{g}}\bar{V}^1 \tag{11.6}$$

Now point (0,1) in a V^1-cell is in fact a vertex and neither V^1 nor V^2 are present in that point. With respect to the matrix this means that the coefficient is "distributed" over neighbouring points, i.e. the points (0,0) and (0,2) for the V^1-component and (1,1) and (-1,1) for the V^2-component. These distributed coefficients form the entries in the matrix.
With respect to the right-hand side of course there is no need for distribution since each row contains only one number.

So the complete process of building the momentum matrix can be written as:

for each velocity component **do**
  for each integration point **do**
    for all "inner" cells **do**
      evaluate the coefficient and store in an array
    end for
    Distribute the array of coefficients over the matrix
    if necessary update the right-hand side
  end for
end for

In the case of the stress tensor the situation is somewhat more complex.
According to the mathematical manual the contribution for the V^1-cell is

$$-\sqrt{g}\mu(g^{11}U^1_{,1} + g^{12}U^1_{,2})|_{(-1,0)}^{(1,0)} - \sqrt{g}\mu(g^{12}U^2_{,1} + g^{22}U^1_{,2})|_{(0,-1)}^{(0,1)} - \{\begin{smallmatrix}1\\ \gamma\beta\end{smallmatrix}\}\tau^{\gamma\beta}\sqrt{g}|_{(0,0)} \tag{11.7}$$

with $U^\alpha_{,\beta}$ given by

$$U^\alpha_{,\beta} = \frac{\partial U^\alpha}{\partial \xi^\beta} + \{\begin{smallmatrix}\alpha\\ \gamma\beta\end{smallmatrix}\}U^\gamma \tag{11.8}$$

and $\tau^{\alpha\beta}$ by

$$\tau^{\alpha\beta} = (\mu + \mu_t)(g^{\alpha\gamma}U^\beta_{,\gamma} + g^{\gamma\beta}U^\alpha_{,\gamma}). \tag{11.9}$$

In this case it is necessary to evaluate the coefficients before the velocity components in the integration points, but also the coefficients before the first derivatives in the computational domain.

These first derivatives require a different distribution stencil, but furthermore they are treated in exactly the same way as the unknowns.

Higher order upwinding is applied by defect correction. This means that with respect to the matrix a first order upwind method is applied, whereas higher order effects are part of the right-hand-side vector.
The first order upwind matrix update is implemented in the distribution procedure only. Hence the computation of the coefficients is independent of upwinding or central differences. In case of central differences the coefficient is distributed equally over neighbouring points, whereas in case of first order upwinding the coefficient is taken in the point at the side where the velocity component originates from. So the sign of the velocity component is taken into account.

11.2.2 Evaluation of the boundary conditions

Once the coefficients for the matrix and right-hand side for the inner cells have been computed, it is necessary to correct for the boundary conditions. From the mathematical guide it is clear that the implementation of the boundary conditions is quite complex. In fact part of the implementation has to be carried out before the distribution of the coefficients over the matrix, whereas another part may be carried out after the matrix has been constructed.

With respect to the momentum equations we have to distinguish between the convection terms and the stress tensor.
The convection terms require only an adaptation for the cells corresponding to the unknowns at the boundary, the so-called half cells.
If the normal velocity is given, the contribution for that row is set equal to zero. Otherwise if the point does not correspond to a multiblock boundary or a boundary with periodical boundary conditions the contributions are recomputed taking into account that only a half cell must be considered.
The stress terms require an adaptation for both the half cells as well as the cells corresponding to points at a half cell distance of the boundary, the so-called tangential cells.
The tangential cell requires an adaptation for the boundary integration point depending on the type of boundary conditions. The normal half cell of course requires a complete update depending on the boundary conditions.
In general multiblock boundaries and boundaries with periodical boundary conditions do not require an update of the inner cell contribution since they are integrated over the boundary. The creation of the virtual cells and the corresponding interpolation must take care of the necessary boundary conditions.

Once the matrix has been constructed in the way described before there are still some elements of stencils near the boundary that are outside the region. These elements are removed by folding them inwardly using linear interpolation and all knowledge of the boundary conditions present. Of course elements corresponding to multiblock boundaries and boundaries with periodical boundary conditions are excluded from this folding process.

11.2.3 Some notation and conventions for the place in the grid

An important disadvantage of staggered grids is the fact that book keeping is more complex than in the case of collocated grids. In the ISNaS code each cell in a block is provided by a i, j, k index, with i from 1 to ni and so on. Points in the cell get the same index, but since a point is part of several cells, it is necessary to make a unique identification. Figure 11.2.5 shows which points in a cell get the same index as the cell itself. This index will be referred to as the global numbering. Since 4 points in R^2 and 8 points in R^3 have the same global number it is necessary to identify these points in a second way. For that reason we have introduced the concept of point type. The global index together with the point type uniquely identifies each point in the grid. The definition of point type is given later on.
Furthermore sometimes we need to know the distance from an integration point to the integration volume centre. This distance is stored in array locdx.
The integration points are numbered in a fixed sequence: the first one is the centre of the cell, the second and third one are in the direction of the cell unknown, starting with the one with the negative distance (of course measured in computational coordinates). The following directions are taken in a cyclic sequence.
Hence in a V^2 cell in R^3 point 1 is the cell centre, point 2 has distance (0,-0.5,0) to the cell centre, point 3 the distance (0,0.5,0), point 4 has distance (0,0,-0.5) and so on.
In some cases we need to know the global index of an integration point. In order to compute this global index easily array dcell has been introduced. This array contains for each integration point the shift of the global index with respect to the global index of the cell centre. Other quantities that are defined in the remainder of this section are acttype, vartype, direction, intn and nintpol.

pnttype The term pnttype (point type) refers to the number assigned to a given point location within a grid cell. In Figure 11.2.5 the sequence numbers for two- and three-dimensional cells are given. The centroid of the cell in R^3 (point number 8) has not been plotted.

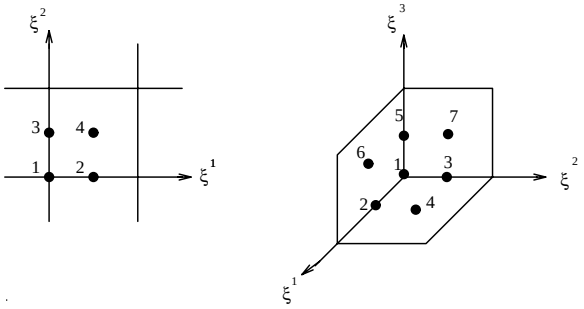

Figure 11.2.5: Sequence number assigned to point location in a grid cell in R^2 and R^3

vartype (variable type) refers to the storage location of a given variable or coefficient. See Figure 11.2.6 for an explanation in R^2.
The following values for *vartype* are possible:

value	R^2	R^3
0	vertex unknown	vertex unknown
1	V^1	V^1
2	V^2	V^2
3	scalar	V^3
4		scalar

acttype defines the action to be taken.
The following values for *action* are possible:

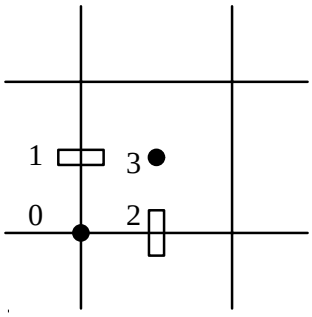

Figure 11.2.6: Location of coefficient or variable in the grid

0 interpolation for point value.
1 finite difference for partial derivative in "1"-direction.
2 finite difference for partial derivative in "2"-direction.
3 finite difference for partial derivative in "3"-direction.

direction Indicates a co-ordinate direction.

intn Indicates an integration point.

nintpol Indicates the number of integration points.

locdx(*direction*,*intn***)** This array of size locdx(1:*ndim*,1:*nintpol*) contains the distance from integration point to cell centre measured in computational co-ordinates and multiplied by two. For example in 2D the number of integration points is 5 (see Figure 11.2.7).
This means that for a V^1 cell in R^2 the array locdx is filled as follows:

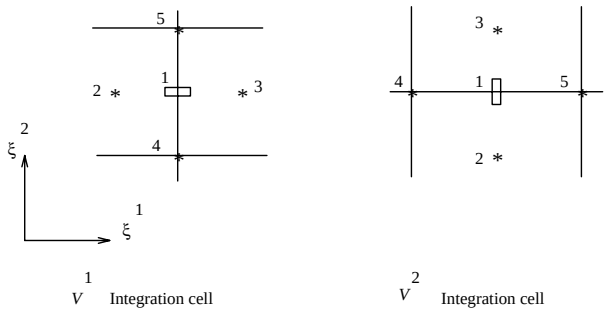

Figure 11.2.7: Integration points internal cells 2D

```
in the 1 direction          in the 2 direction

locdx(1,1)=  0              locdx(2,1)=  0
locdx(1,2)= -1              locdx(2,2)=  0
locdx(1,3)=  1              locdx(2,3)=  0
locdx(1,4)=  0              locdx(2,4)= -1
locdx(1,5)=  0              locdx(2,5)=  1
```

And for a V^2 cell:

```
in the 1 direction          in the 2 direction

locdx(1,1)=  0              locdx(2,1)=  0
```

```
locdx(1,2)=  0          locdx(2,2)= -1
locdx(1,3)=  0          locdx(2,3)=  1
locdx(1,4)= -1          locdx(2,4)=  0
locdx(1,5)=  1          locdx(2,5)=  0
```

dcell(*direction*,*intn***)** This array of size dcell(1:3,1:numpol) contains for each integration point the shift of the cell index of this integration point with respect to the cell containing the centre of the integration cell.
This means that for a V^1 cell in R^2 the array dcell is filled as follows:

```
in the 1 direction          in the 2 direction

dcell(1,1)=  0              dcell(2,1)=  0
dcell(1,2)= -1              dcell(2,2)=  0
dcell(1,3)=  0              dcell(2,3)=  0
dcell(1,4)=  0              dcell(2,4)=  0
dcell(1,5)=  0              dcell(2,5)=  1
```

And for a V^2 cell:

```
in the 1 direction          in the 2 direction

dcell(1,1)=  0              dcell(2,1)=  0
dcell(1,2)= -1              dcell(2,2)=  0
dcell(1,3)=  0              dcell(2,3)=  0
dcell(1,4)=  0              dcell(2,4)=  0
dcell(1,5)=  1              dcell(2,5)=  0
```

11.2.4 Interpolation and finite difference formulae

Interpolations and finite difference formula are so simple that usually they are explicitly given in the code when needed. However, there is one exception. In the distribution of the momentum equations we need to know the matrix entries and especially the position of the matrix entries in the actual matrix storage. The matrix itself is stored as a two-dimensional matrix with the same number of rows as there are unknowns. The number of columns actually stored is identical to the number of items in the stencil. The diagonal sequence number (or column number) of each entry in the stencil is given in the matrix information arrays mct treated in Section 11.2.5. In order to find the corresponding column number from these arrays it is necessary to know the distances from the interpolation points to the integration cell centre. For that reason two arrays are introduced: *nintpol* containing the number of interpolation points and *dij* containing the corresponding distances.
With respect to the stress tensor the distribution process not only involves interpolation, but also differentiation. This is also automized using two extra arrays *nintder* containing the number of differentiation points and *didj* containing the corresponding distances.

nintpol(*ndim***)** This array of size *ndim* defines the number of points used in a interpolation formula for a given integration point and a given integration cell direction. Suppose we consider a cell corresponding to a V^α unknown, then $nintpol(\alpha)$ defines the number of interpolation points needed for a V^α unknown and $nintpol(\beta)$ for a V^β unknown, where β and α are different.

dij(*direction*,*interpolation_point*,*direction***)** Array containing the distances of the interpolation points to the cell centra in global co-ordinates dij(α,i,j) denotes the distance of the i-th interpolation point, component j and with respect to the V^α unknown.
With respect to the interpolation we have the following conventions:

Consider a V1 cell with 5 integration points
The coefficient of a velocity coefficient in the equation at the integration point is given. Since the velocity does not have to be present at this integration point it is necessary to distribute the coefficient over surrounding neighbours. For example V^2 in a V^1 point (pnttyp=3) is created by the linear combination: (V^2(i,j)+V^2(i,j+1)+V^2(i-1,j)+V^2(i-1,j+1))/4
The number of distribution points is stored in nintpol.
The positions in the row of the matrix will be called connections They are computed using the array mct.
Array mct depends on alphad, the local direction, the point type and also of the position of the point relative to the cell centre. With respect to the V alphad coefficient the position is equal to the local co-ordinate divided by two.
In R^2 we need the following table with respect to the direction perpendicular α

```
Local co-ordinate      V1              V2

 (-1,-1)            (-1, 0)         ( 0,-1)
 ( 1,-1)            ( 0, 0)         ( 1,-1)
 (-1, 1)            (-1, 1)         ( 0, 0)
 ( 1, 1)            ( 0, 1)         ( 1, 0)
```

The distribution in R^2 is given by the following table:

```
alphad  locdx  pnttyp betad  nintpol  local co-ordinates:

   1   (0,0)   3     1       1       (0,0)
                     2       4       (+/- 1, +/- 1)
     (+/-1,0)  4     1       2       (0,0), locdx
                     2       2       (locdx(1), +/- 1)
     (0,+/-1)  1     1       2       (0,0), locdx
                     2       2       (+/- 1,locdx(2))

   2   (0,0)   2     2       1       (0,0)
                     1       4       (+/- 1, +/- 1)
     (+/-1,0)  1     2       2       (0,0), locdx
                     1       2       (locdx(1), +/- 1)
     (0,+/-1)  4     2       2       (0,0), locdx
                     1       2       (+/- 1,locdx(2))
```

nintder(*direction*,*direction***)** This array of size $ndim \times ndim$ defines the number of points used in a differentiation formula for a given integration point and a given integration cell direction. Suppose we consider a cell corresponding to a V^α unknown, then $nintpol(\alpha, \gamma)$ defines the number of points needed for a $\frac{\partial V^\alpha}{\partial \xi^\gamma}$ derivative.

didj(*direction*,*direction*,*interpolation_point*,*direction***)** Array containing the distances of the finite difference points to the cell centra in global co-ordinates didj(α, β, i, j) denotes the distance of the i-th finite difference point, component j and with respect to the $\frac{\partial V^\alpha}{\partial \xi^\gamma}$ derivative. The differentiation in R^2 is given by the following table:

```
V1-cell:  dU^alpha/dksi^beta

locdx      alpha,beta    n   fact            (di,dj)
```

```
 (0,0)       1   1        2  1/2(1,-1)       (1,0) (-1,0)
             1   2        2  1/2(1,-1)       (0,1) (0,-1)
             2   1        4  1/2(1,-1,1,-1)  (1,1) (-1,1) (1,-1) (-1,-1)
             2   2        4  1/2(1,-1,1,-1)  (1,1) (1,-1) (-1,1) (-1,-1)

(-1,0)       1   1        2     (-1,1)       (-1,0) (0,0)
             1   2        4  1/4(1,-1,1,-1)  (-1,1)(-1,-1) (0,1) (0,-1)
             2   1        0
             2   2        2     (1,-1)       (-1,1) (-1,-1)

 (1,0)       1   1        2     (1,-1)       (1,0)  (0,0)
             1   2        4  1/4(1,-1,1,-1)  (1,1) (1,-1) (0,1) (0,-1)
             2   1        0
             2   2        2     (1,-1)       ( 1,1) ( 1,-1)

(0,-1)       1   1        4  1/4(1,-1,1,-1)  (1,-1)(-1,-1) (1,0) (-1,0)
             1   2        2     (-1,1)       (0,-1) (0,0)
             2   1        2     (1,-1)       (1,-1) (-1,-1)
             2   2        0

(0,1)        1   1        4  1/4(1,-1,1,-1)  (-1,1)(1,1) (-1,0) (1,0)
             1   2        2     (1,-1)       (0,1) (0,0)
             2   1        2     (1,-1)       (1,1) (-1,1)
             2   2        0

 V2-cell:  dU^alpha/dksi^beta

 locdx      alpha,beta    n   fact             (di,dj)

 (0,0)       2   2        2  1/2(1,-1)       (0,1) (0,-1)
             2   1        2  1/2(1,-1)       (1,0) (-1,0)
             1   2        4  1/2(1,-1,1,-1)  (1,1) (1,-1) (-1,1) (-1,-1)
             1   1        4  1/2(1,-1,1,-1)  (1,1) (-1,1) (1,-1) (-1,-1)

(0,-1)       2   2        2     (-1,1)       (0,-1) (0,0)
             2   1        4  1/4(1,-1,1,-1)  (1,0) (-1,0) (1,-1)(-1,-1)
             1   2        0
             1   1        2     (1,-1)       (1,-1) (-1,-1)

 (0,1)       2   2        2     (1,-1)       (0,1)  (0,0)
             2   1        4  1/4(1,-1,1,-1)  (1,1) (-1,1) (1,0) (-1,0)
             1   2        0
             1   1        2     (1,-1)       ( 1,1) ( -1,1)

(-1,0)       2   2        4  1/4(1,-1,1,-1)  (-1,1)(-1,-1) (0,1) (0,-1)
             2   1        2     (-1,1)       (-1,0) (0,0)
             1   2        2     (1,-1)       (-1,1) (-1,-1)
             1   1        0

(1,0)        2   2        4  1/4(1,-1,1,-1)  (1,1)(1,-1) (0,1) (0,-1)
             2   1        2     (1,-1)       (1,0) (0,0)
             1   2        2     (1,-1)       (1,1) (1,-1)
             1   1        0
```

cv(*direction*,*direction*,*interpolation_point***)** This array defines the weights for the interpolation or finite difference formula. It is in correspondence with the didj table.

11.2.5 The matrix information arrays

Integer information about the matrices is stored in these arrays. For each matrix three different kind of information arrays are stored.

For all matrix manipulations the information about the matrices should be retrieved from these arrays. In this way there is only one place where the information about the matrices is stored.

First, the matrix code table (**mct**) is stored. This table translates a given variable type and shift to diagonal number. These diagonals are numbered from 1 to nconct (total number of connections).

```
Momentum matrix
mct(1:ndim,1:ndim,-maxdx:maxdx,-maxdx:maxdx,-maxdx:maxdx)

Gradient matrix
mct(1:ndim,1:1,-maxdx:maxdx,-maxdx:maxdx,-maxdx:maxdx)

Transport matrix
mct(1:1,1:1,-maxdx:maxdx,-maxdx:maxdx,-maxdx:maxdx)

with :
  ndim  = dimension of space
  maxdx = maximum relative shift between cell indices
          and variable indices

usage :

  alpha = 1...ndim, beta = 1...ndim
     Matrix(i,j,k,alpha,mct(alpha,beta,di,dj,dk))
     is the matrix element in row i,j,k,alpha
     of the momentum matrix corresponding
     to V(beta,i+di,j+dj,k+dk)

  alpha = 1...ndim, beta = ndim+1
     Matrix(i,j,k,alpha,mct(alpha,beta,di,dj,dk))
     is the matrix element in row i,j,k,alpha
     of the momentum matrix corresponding
     to Pressure(i+di,j+dj,k+dk)

  alpha = ndim+1, beta = ndim+1
     Matrix(i,j,k,alpha,mct(alpha,beta,di,dj,dk))
     is the matrix element in row i,j,k
     of the divgrad matrix corresponding
     to Pressure(i+di,j+dj,k+dk)
```

Second, the matrix code table inverse (**mctinv**) is stored. This table translates a diagonal number to a variable type and shift.

```
Momentum matrix
```

```
mctinv(1:maxdim,1:ndim,1:nconct)

Gradient matrix
mctinv(1:maxdim,1:ndim,1:nconct)

Transport matrix
mctinv(1:maxdim,1:ndim,1:nconct)


with :
  ndim   = dimension of space
  nconct = number of connections
  maxdim = maximum dimension of space

usage :

  Momentum matrix

  alpha = 1...ndim, beta = 1...ndim
     xcode = mct(alpha,beta,di,dj,dk)
     mctinv(0,alpha,xcode) = beta
     mctinv(1,alpha,xcode) = di
     mctinv(2,alpha,xcode) = dj
     mctinv(3,alpha,xcode) = dk

  Pressure matrix

  alpha = 1...ndim, beta = 1,1
     xcode = mct(alpha,beta,di,dj,dk)
     mctinv(0,alpha,xcode) = beta
     mctinv(1,alpha,xcode) = di
     mctinv(2,alpha,xcode) = dj
     mctinv(3,alpha,xcode) = dk

  Divgrad (Transport) matrix

  alpha = 1,1  beta = 1,1
     xcode = mct(alpha,beta,di,dj,dk)
     mctinv(0,alpha,xcode) = 1
     mctinv(1,alpha,xcode) = di
     mctinv(2,alpha,xcode) = dj
     mctinv(3,alpha,xcode) = dk
```

Third, the matrix shift table (**mst**) is stored. This table translates a given diagonal number to a relative column number.

```
Momentum matrix
mst(1:ndim,1:nconct)

Gradient matrix
mst(1:ndim,1:nconct)

Transport matrix
```

```
mst(1:1,1:nconct)


with :
  ndim   = dimension of space
  nconct = number of connections

usage :

     mst(alpha,xcode) =
         di+dj*(ni+1)+dk*(ni+1)*(nj+1)+
         (beta-alpha)*(ni+1)*(nj+1)*(nk+1)
```

11.3 Reference structure

The general matrix assembly subroutine is subroutine ISBUILD. This subroutine builds the matrix and right-hand side for all blocks at one node. The structure of this subroutine and its main under subroutines is given in the following subsections. The subroutine is called for each time-step.

11.3.1 Reference structure of the general matrix assembly subroutine

```
isbuild                          General matrix builder

   islsds01                      Compute length of local arrays

   isfilm                        fill integer information about matrices

   isfilb                        fill boundary values

   issysm                        fill momentum equations

   issysp                        fill pressure equations

   issyst                        fill transport equations
```

11.3.2 Reference structure of the momentum matrix assembly subroutine

The momentum building subroutine builds the momentum matrix and right-hand side including the pressure gradient. For the building of these parts two subroutines ismom2d and ismom3d are used. However, in the case of the accurate Wesseling and van Beek discretization we use some old fashioned subroutines that still have to be adapted. In the future these subroutines will be removed. Extra options in in isbuild are for example the printing of matrix and right-hand side at request.

The structure of subroutine ismom2d is as follows:

```
   issetcon                      Clear matrix and right hand side
```

```
iscova2d                        Compute covariant basis functions
For alphad := 1, 2 do           2 components
   isintcf2                     interpolate rho
   ismomtd2                     Build time-derivative
   isbmomt2                     Correct for half cells
   For itn := 1 (1) 5 do        5 integration points
      ismakmp2                  Compute local co-ordinates
      isjcta2d                  Compute contra variant base vectors
      isjctg2d                  Compute metric tensor
      ischrys2                  Compute christoffel symbols
      ismomcv2                  Compute coefficients of convective terms
      isbmomc2                  correct for half cells
      isdisc2d                  Distribute
      ismomct2                  Compute coefficients of stress terms
      isbmoms1                  correct for tangential cells
      isbmoms2                  correct for half cells
      isdiss2d                  Distribute
iscova2d                        Compute covariant basis functions
```

Chapter 12

Time integration

Chapter 13

The multiblock process

Chapter 14

General tools

14.1 Introduction

14.2 Copy subroutines

Chapter 15

Debugging possibilities

15.1 Introduction

At this moment the number of debug possibilities of ISNaS is rather limited. In fact there are two debug possibilities available.
First of all the memory management can be checked by using the file isnas.dbg as descibed in Section 2.2.5.
Besides that, it is possible to print matrices, right-hand side vectors and solution vectors during the computation. See the Users Manual for a description.

Chapter 16

Data structures

16.1 introduction

In this chapter the following abbreviations are used:

NBLOCK number of blocks in a multi-block algorithm

NDIM dimension of space (2 or 3)

NI Number of non-virtual cells in the i-direction (one block)

NJ Number of non-virtual cells in the j-direction (one block)

NK Number of non-virtual cells in the k-direction (one block)

NPOINT Number of vertices of non-virtual cells in one block, i.e. (NI+1)(NJ+1)(NK+1)

NVIRTUAL_rows Number of rows of virtual cells in the local co-ordinates array. This item is also indicated by NVIRTUAL.

NPOINT_EXT Number of vertices of all cells in one block including the virtual ones, i.e. (NI+1+2NVIRTUAL)(NJ+1+2NVIRTUAL)(NK+1+2NVIRTUAL)

NTIME_LEVEL Number of time levels or perhaps iterations that must be stored in the global solution arrays

NTIME_LEVEL_local Number of time levels or perhaps iterations that must be stored in the local solution arrays

NDEGFD Number of degrees of freedom
In 2D these are u, v, p and the scalar quantities, in 3D u, v, w, p and the scalar quantities.

NSOLUT Number of unknowns for one-degree of freedom in one block

NCONCT Number of connections in one row of the matrix.

NUSOL Number of unknowns for coupled degrees of freedom in one block

NTRANS Number of transport equations including the turbulence equations

NTRNSP Number of transport equations without the turbulence equations

NTURB Number of turbulence equations

NMACHINES Number of computers the program is running

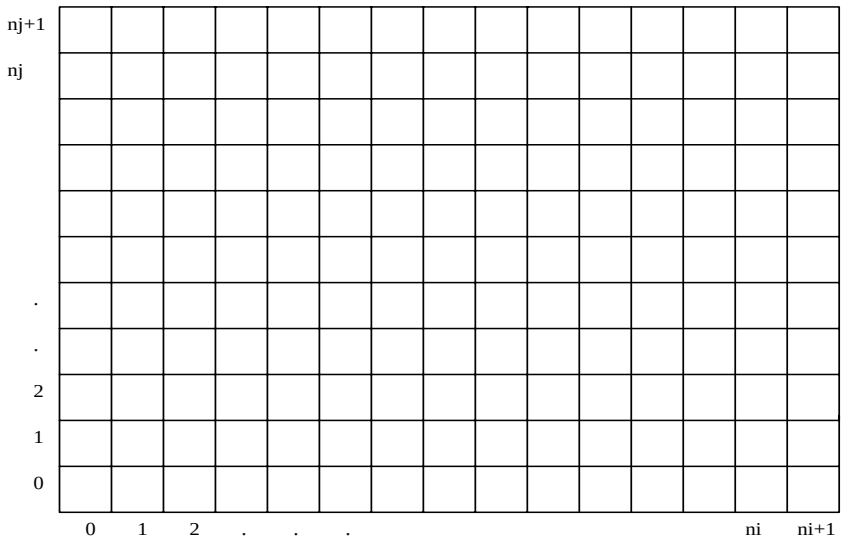

Figure 16.1.1: 2D computational grid

The 2D computational grid has the shape as sketched in Figure 16.1.1 per block:

Each cell gets an i- and a j-number. The cells with i-number 0 or ni+1 and the cells with j-number 0 and nj+1 are virtual cells which are not used for the computation.

Within cell (i,j) the numbering given in Figure 16.1.2 is used for the variables and nodes.

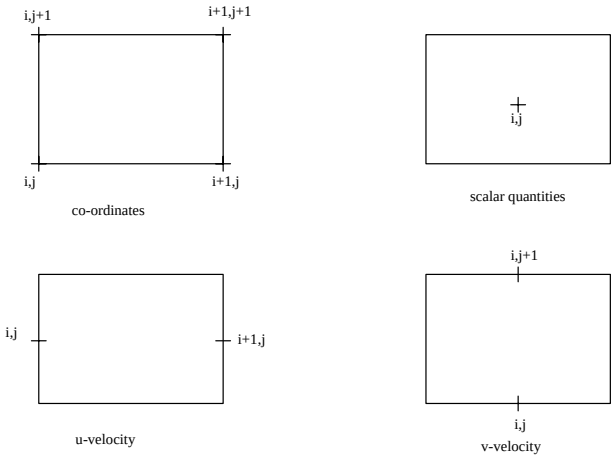

Figure 16.1.2: Numbering of variables in cells

In 3D the same construction and numbering is applied. This means that for cell i,j,k the co-ordinates have index numbers i+a,j+b,k+c with a,b,c = 0 or 1. The U1-velocity component has indices i+a,j,k. The U2-velocity component has indices i,j+b,k. The U3-velocity component has indices i,j,k+c. The scalar component has indices i,j,k

In this chapter the complete description of the arrays and the general common blocks is given. It concerns the following arrays and commons:

coor 16.3	Contains the co-ordinates of the grid points.
solut 16.4	Contains the solution array.
matrix 16.6	Contains the momentum matrix or the transport matrix.
dgmat 16.6	Contains the pressure matrix.
gmat 16.6	Contains the gradient matrix.
rhside 16.7	Contains the right-hand side vector.
infmm 16.8	Contains integer information about the storage of the momentum matrix.
infmg 16.8	Contains integer information about the storage of the pressure gradient matrix.
infmg 16.8	Contains integer information about the storage of the pressure and transport matrix.
iinput 16.9	Contains integer user input.
rinput 16.9	Contains real user input.
iaction 16.18.6	Contains information about the actions that must be carried out by the node program.

16.2 Sequence of the unknowns

At this moment the sequence of the unknowns as stored in the solution array, the boundary conditions arrays and so-on is fixed. In the future it is possible that this sequence may be changed. The sequence of the storage may be different from the sequence in which the equations are solved. The following sequence is used:

1 Velocity vector (Contravariant components)

2 Pressure (scalar defined in centroids)

3-2+NTRNSP Transport unknowns (Scalar defined in centroids)

3+NTRNSP-2+NTRNSP+NTURB Turbulence unknowns (Scalar defined in centroids)

In case of turbulence equations the NTURB turbulence unknowns are defined as follows:

$k-\varepsilon$**-model** 1 turbulent kinetic energy k

2 dissipation rate of turbulent kinetic energy per unit mass ε

$k-\omega$**-model** 1 turbulent kinetic energy k

2 specific dissipation rate of turbulent kinetic energy per unit mass ω

In case of compressible flow the sequence of the unknowns at present is:

1 Velocity vector

2 Pressure

3 enthalpy h

16.3 The co-ordinates arrays

The reference with respect to the buffer array of the co-ordinates arrays is stored in an integer array ISCOOR of size NBLOCK. This array is available at the host. At the nodes the same type of array is available, however, if block IBLOCK is not available at the node ISCOOR(IBLOCK) has the value 0.

The corresponding co-ordinates array may be considered as a two- dimensional array of size NPOINT_EXT $\times$ NDIM:

COOR (1 : NPOINT_EXT, 1 : NDIM)

with:

NPOINT_EXT : Number of vertices of all cells
NPOINT_EXT = (NI+1+2∗NVIRTUAL) ∗ (NJ+1+2∗NVIRTUAL) (2D)
NPOINT_EXT = (NI+1+2∗NVIRTUAL) ∗ (NJ+1+2∗NVIRTUAL) ∗ (NK+1+2∗NVIRTUAL) (3D)

NDIM : Dimension of space

The co-ordinates are numbered such that the "i-direction" increases the fastest and the "k-direction" the slowest.

Locally array COOR may be considered as a three-dimensional array (2D) or a four-dimensional array (3D) as follows:

2D : Double precision three-dimensional array.

COOR (i,j,l) l=1,2 ; i=1-nvirtual(1)ni+1+nvirtual; j=1-nvirtual(1)nj+1+nvirtual.

COOR (i,j,1) x-coordinate of vertex (i,j), being the left-under point of cell(i,j).

COOR (i,j,2) y-coordinate of vertex(i,j).

Mapping from 2D array to 3D array is as follows:

COOR (2D) ((j-1)∗(ni+1+2∗nvirtual)+i,l) = COOR (3D) (i,j,l)

3D : Double precision four-dimensional array.

COOR (i,j,k,l) l=1,2,3; i=1-nvirtual(1)ni+1+nvirtual; j=1-nvirtual(1)nj+1+nvirtual; k=1-nvirtual(1)nk+1+nvirtual.

COOR (i,j,k,1) x-coordinate of vertex(i,j,k),

COOR (i,j,k,2) y-coordinate of vertex(i,j,k).

COOR (i,j,k,3) z-coordinate of vertex(i,j,k).

Mapping from 2D array to 4D array is as follows:

COOR (2D) ((k-1)∗(ni+1+2∗nvirtual) ∗(nj+1+2∗nvirtual)+(j-1)∗(ni+1+2∗nvirtual)+i,l) = COOR (4D) (i,j,k,l)

16.4 The solution arrays

The reference with respect to the buffer array of the solution arrays is stored in an integer array ISSOLUT of size NBLOCK. This array is available at the host. At the nodes the same type of array is available, however, if block IBLOCK is not available at the node ISSOLUT(IBLOCK) has the value 0.

In order to get equal sized arrays the solution arrays are extended with dummies. So each solution vector has ni+1+2nvirtual points in the i-direction nj+1+2nvirtual points in the j-direction and nk+1+2nvirtual points in the k-direction. Therefore the solution array may be considered as a double precision four dimensional(2D) or five dimensional (3D) array.

2D:

SOLUT (i, j, l, time) with
i = 1-nvirtual (1) ni+1+nvirtual
j = 1-nvirtual (1) nj+1+nvirtual
l = 1-nvirtual (1) NDEGFD
time = 1, 2, ... , ntime_level

i,j position of unknown in the grid

l sequence number of unknown

time one of the time levels

3D:

SOLUT (i, j, k, l, time) with
i = 1 (1) ni+1+nvirtual
j = 1-nvirtual (1) nj+1+nvirtual
k = 1-nvirtual (1) nk+1+nvirtual
l = 1-nvirtual (1) NDEGFD
time = 1, 2, ... , ntime_level

i, j, k position of unknown in the grid

Locally SOLUT may be treated as a three-dimensional array of length NSOLUT x NDEGFD x NTIME_LEVEL. In that case
SOLUT(i,j,k,l,time) =
SOLUT_1D(i+(j-1)*(NI+1+2*NVIRTUAL)+ (k-1)*(NI+1+2*NVIRTUAL)*(NJ+1+2*NVIRTUAL),l,time)

16.5 Geometrical quantities

The geometrical quantities are only stored in the one-block solvers. These arrays are treated as temporary arrays, which means that they are created in the one-block solver and destroyed if the next block or step is considered. At present the storage is restricted to the storage of sqrt(g) and 1/sqrt(g) in all points of the grid. Exactly the same cells as for the co-ordinates are used.
In case of a turbulent flow, also the wall distance with respect to no-slip walls is stored. This part can only be used in combination with array IGEOM.

Locally, the first two parts of array GEOM may be considered as a three-dimensional array (2D) or a four-dimensional array (3D) as follows:

2D: Double precision four-dimensional array.

GEOM(i,j,l,m) with
i = 1-NVIRTUAL_rows (1) ni+1+NVIRTUAL_rows
j = 1-NVIRTUAL_rows (1) nj+1+NVIRTUAL_rows
l = 1 (1) 4
m = 1 (1) 2

i,j denotes the cell number
m denotes the quantity to be stored:

m =1: $\sqrt{g}$

m =2: $\frac{1}{\sqrt{g}}$

l denotes the sequence number in the cell according to the scheme in Figure 16.5.3.

3D: Double precision five-dimensional array.

GEOM(i,j,k,l,m) with
i = 1-NVIRTUAL_rows (1) ni+1+NVIRTUAL_rows
j = 1-NVIRTUAL_rows (1) nj+1+NVIRTUAL_rows
k = 1-NVIRTUAL_rows (1) nk+1+NVIRTUAL_rows
l = 1 (1) 8
m = 1 (1) 2

i,j,k denotes the cell number
m denotes the quantity to be stored:

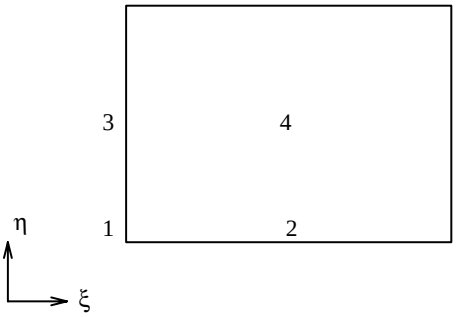

Figure 16.5.3: positions in P-cells

m =1: $\sqrt{g}$

m =2: $\frac{1}{\sqrt{g}}$

l denotes the sequence number in the cell according to the following scheme:

l=1: shift 0,0,0 with respect to vertex i,j,k
l=2: shift 0.5,0,0 with respect to vertex i,j,k
l=3: shift 0,0.5,0 with respect to vertex i,j,k
l=4: shift 0.5,0.5,0 with respect to vertex i,j,k
l=5: shift 0,0,0.5 with respect to vertex i,j,k
l=6: shift 0.5,0,0.5 with respect to vertex i,j,k
l=7: shift 0,0.5,0.5 with respect to vertex i,j,k
l=8: shift 0.5,0.5,0.5 with respect to vertex i,j,k

In case of turbulent flow, array GEOM is extended with an extra array of length NSOLUT, with NSOLUT the number of entries in array SOLUT for each degree of freedom. In this third array the wall distance with respect to the closest point at the closest no-slip wall is stored.
The wall distance is stored in the cell centroids??? Fred is this correct???

Array IGEOM is used if array GEOM part 3 is filled, i.e. in case of turbulent flow. The length of this integer array is also equal to NSOLUT. IGEOM contains FRED???????

16.6 The matrices

The matrices to be used in the ISNaS program are all stored temporarily. They are only used in the one-block solvers. So in each time-step or iteration the matrices are built again. However, there is one exception. If during the multi block iteration enough space is available it might be possible to store the matrix for a particular equation like for example the momentum equation. The reason is that the same matrix is used in several multi block iterations. Such a storage makes only sense if the matrix can be kept in-core together with all other vectors to be used in the multi block process.

Depending on the solution method the following matrices may be used:

MATRIX : This is either the momentum matrix (coupled or decoupled) or a transport matrix

DGMAT : This is the pressure matrix for the pressure-correction method

GMAT : This is the discretization matrix corresponding to the gradient of the pressure

Since we allow storage of the matrices for several blocks the sequence variables ISMATRIX, ISDGMAT and ISGMAT all get the dimension NBLOCKS, although in an application it might be possible that only the first entry is used.

The following storage is used for the various matrices.

- The momentum matrix (decoupled situation)

  In this case the matrix relates to all velocity unknowns

  The matrix may be considered as a two-dimensional double precision array of size NCONCT ∗ NUSOL in which the matrix is stored.

  Storage scheme:

  Three-dimensional double precision array of size NCONCT ∗ NUSOL ∗ NDIM in which the matrix is stored. Information about the shifts of the off-diagonal-terms is stored in array INFMM

  matrix(1:ni+1, 1:nj+1, 1:ndim, 1:nconct),

  In 2D the storage of Figure 16.6.4 is used: The sequence of the unknowns corresponds to

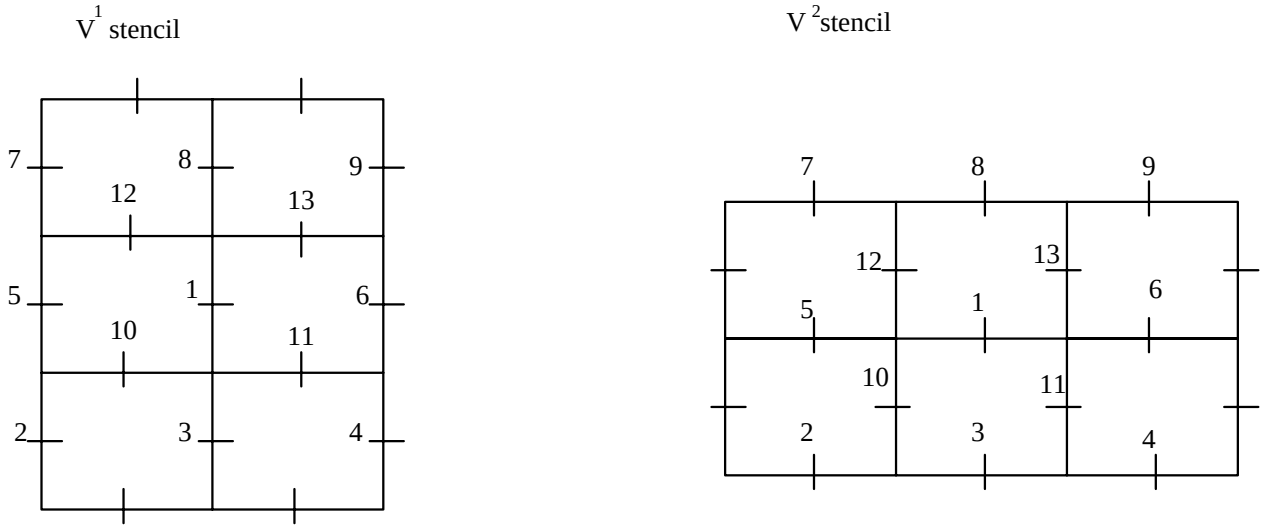

Figure 16.6.4: Velocity stencils in 2D

the sequence used in SOLUT

In 3D the following storage is used:

Since in this case it is hardly impossible to draw the connections, we give the connections in formulae. We define the shifts with respect to the diagonal

```
row   =  ni+1
plane = (ni+1)*(nj+1)
cube  = (ni+1)*(nj+1)*(nk+1)
```

In Figure 16.6.5 the V^1 connections in 3D are sketched.

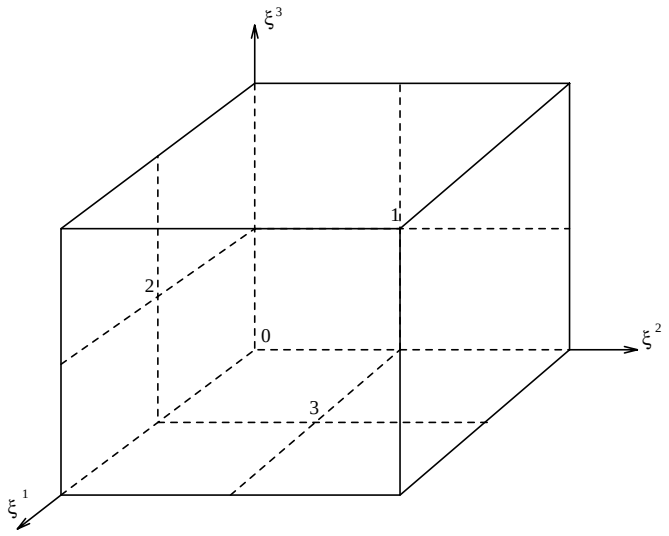

Figure 16.6.5: V^1 connections in 3D

```
                  1
*** connections V      3D  ( shifts )
 2              1
V     w.r.t   V     +1, -1, 0   units half a cell +cube ( nr.27 )

 3              1
V     w.r.t   V     +1,  0, -1  units half a cell + 2*cube ( nr.41 )

1  :  0
2  : -1  -row
3  :     -row
4  : +1  -row
5  : -1
6  : +1
7  : -1  +row
8  :     +row
9  : +1  +row
10 :     -row   -plane
11 : -1         -plane
12 :            -plane
13 : +1         -plane
14 :     +row   -plane
15 :     -row   +plane
16 : -1         +plane
17 :            +plane
18 : +1         +plane
19 :     +row   +plane

20 : -1         -plane  +cube
21 :            -plane  +cube
22 : -1   +row  -plane  +cube
23 :      +row  -plane  +cube
24 : -1   -row          +cube
```

```
25 :      -row          +cube
26 : -1                 +cube
27 :                    +cube
28 : -1   +row          +cube
29 :      +row          +cube
30 : -1 +2*row          +cube
31 :    +2*row          +cube
32 : -1         +plane  +cube
33 :            +plane  +cube
34 : -1   +row  +plane  +cube
35 :      +row  +plane  +cube

36 : -1         -plane  +2*cube
37 :            -plane  +2*cube
38 : -1   -row          +2*cube
39 :      -row          +2*cube
40 : -1                 +2*cube
41 :                    +2*cube
42 : -1   +row          +2*cube
43 :      +row          +2*cube
44 : -1   -row  +plane  +2*cube
45 :      -row  +plane  +2*cube
46 : -1         +plane  +2*cube
47 :            +plane  +2*cube
48 : -1   +row  +plane  +2*cube
49 :      +row  +plane  +2*cube
50 : -1       +2*plane  +2*cube
51 :           2*plane  +2*cube


                 2
*** connections V      3D  ( shifts )

 3            2
V     w.r.t   V      0, +1, -1   units half a cell +cube ( nr.26 )

 1            2
V     w.r.t   V     -1, +1,  0   units half a cell -cube ( nr.45 )

1  :  0
2  : -1  -row
3  :     -row
4  : +1  -row
5  : -1
6  : +1
7  : -1  +row
8  :     +row
9  : +1  +row
10 :     -row   -plane
11 : -1         -plane
12 :            -plane
13 : +1         -plane
14 :     +row   -plane
15 :     -row   +plane
```

```
16 : -1         +plane
17 :            +plane
18 : +1         +plane
19 :     +row   +plane

20 :     -row   -plane  +cube
21 :            -plane  +cube
22 : -1  -row           +cube
23 :     -row           +cube
24 : +1  -row           +cube
25 : -1                 +cube
26 :                    +cube
27 : +1                 +cube
28 : -1   -row  +plane  +cube
29 :      -row  +plane  +cube
30 : +1   -row  +plane  +cube
31 : -1         +plane  +cube
32 :            +plane  +cube
33 : +1         +plane  +cube
34 :      -row+2*plane  +cube
35 :           2*plane  +cube

36 :      -row  -plane  -cube
37 : +1   -row  -plane  -cube
38 :            -plane  -cube
39 : +1         -plane  -cube
40 : -1   -row          -cube
41 :      -row          -cube
42 : +1   -row          -cube
43 : +2   -row          -cube
44 : -1                 -cube
45 :                    -cube
46 : +1                 -cube
47 : +2                 -cube
48 :      -row  +plane  -cube
49 : +1   -row  +plane  -cube
50 :            +plane  -cube
51 : +1         +plane  -cube


                3
*** connections V      3D  ( shifts )

 1            3
V     w.r.t   V     -1,  0, +1  units half a cell - 2*cube ( nr.31 )

 2            3
V     w.r.t   V      0, -1, +1  units half a cell - cube ( nr.46 )

1  :  0
2  : -1  -row
3  :     -row
4  : +1  -row
5  : -1
```

```
6  : +1
7  : -1  +row
8  :     +row
9  : +1  +row
10 :     -row   -plane
11 : -1         -plane
12 :            -plane
13 : +1         -plane
14 :     +row   -plane
15 :     -row   +plane
16 : -1         +plane
17 :            +plane
18 : +1         +plane
19 :     +row   +plane


20 :     -row   -plane  -2*cube
21 : +1  -row   -plane  -2*cube
22 : -1         -plane  -2*cube
23 :            -plane  -2*cube
24 : +1         -plane  -2*cube
25 : +2         -plane  -2*cube
26 :      +row  -plane  -2*cube
27 : +1   +row  -plane  -2*cube
28 :      -row          -2*cube
29 : +1   -row          -2*cube
30 : -1                 -2*cube
31 :                    -2*cube
32 : +1                 -2*cube
33 : +2                 -2*cube
34 :      +row          -2*cube
35 : +1   +row          -2*cube

36 :      -row  -plane  -cube
37 : -1         -plane  -cube
38 :            -plane  -cube
39 : +1         -plane  -cube
40 : -1   +row  -plane  -cube
41 :      +row  -plane  -cube
42 : +1   +row  -plane  -cube
43 :    +2*row  -plane  -cube
44 :      -row          -cube
45 : -1                 -cube
46 :                    -cube
47 : +1                 -cube
48 : -1   +row          -cube
49 :      +row          -cube
50 : +1   +row          -cube
51 :    +2*row          -cube
```

- The transport matrix Two-dimensional array of size 9∗NUSOL (2D) in which the pressure matrix is stored.
  Storage scheme:

Three-dimensional double precision array of size NCONCT ∗ NUSOL ∗ NDIM in which the matrix is stored. Information about the shifts of the off-diagonal-terms is stored in array INFMT.

In 2D the stencil is given by:

matrix(1:ni+1, 1:nj+1, 1:nconct), see Figure 16.6.6 The sequence of the unknowns corre-

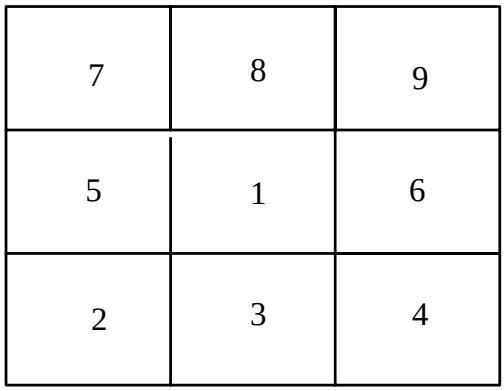

Figure 16.6.6: stencil for scalars

sponds to the sequence used in SOLUT

The 3D stencil is too complicated to plot so we use the same method as for the momentum matrix

```
1  :  0
2  : -1  -row
3  :     -row
4  : +1  -row
5  : -1
6  : +1
7  : -1  +row
8  :     +row
9  : +1  +row
10 :     -row -plane
11 : -1       -plane
12 :          -plane
13 : +1       -plane
14 :     +row -plane
15 :     -row +plane
16 : -1       +plane
17 :          +plane
18 : +1       +plane
19 :     +row +plane
```

- The pressure matrix (DGMAT)

  The storage of the pressure matrix is identical to the storage of the transport matrices

- The gradient matrix (GMAT)

Storage scheme :

Three-dimensional double precision array of size NCONCT ∗ NUSOL ∗ NDIM in which the matrix is stored. Information about the shifts of the off-diagonal-terms is stored in array INFMG

In 2D the storage is as follows:

gmat(1:ni+1, 1:nj+1, 1:ndim, 1:nconct), see Figure 16.6.7

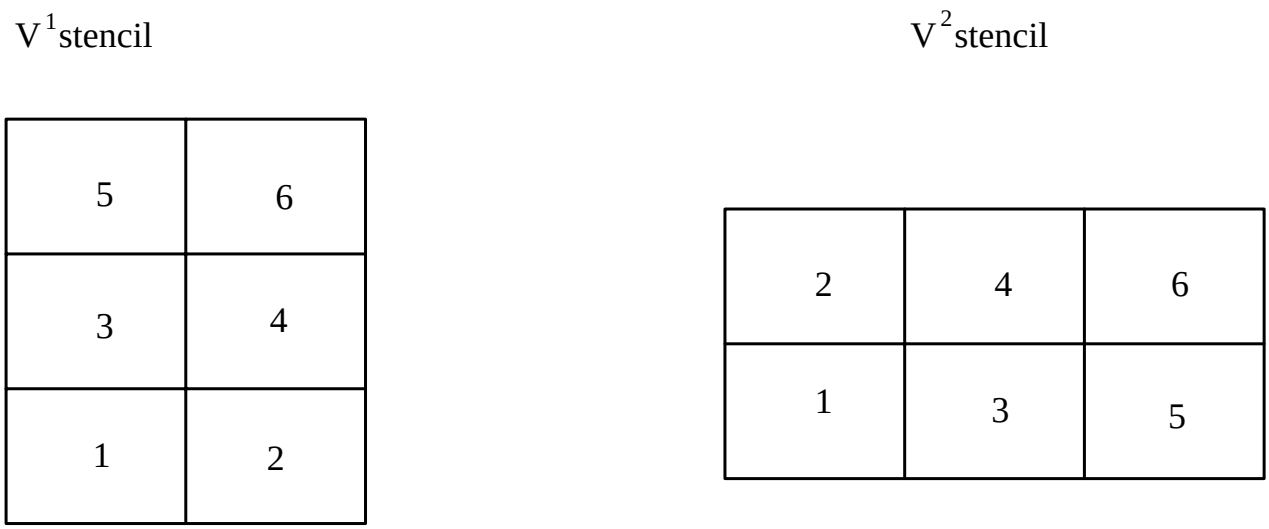

Figure 16.6.7: stencil for G-matrix

In 3D the storage is given in the same way as for the momentum matrix

```
          1
*** G    V

1  : -1  -row
2  :     -row
3  : -1
4  :  0
5  : -1  +row
6  :      row
7  : -1       -plane
8  :           -plane
9  : -1       +plane
10 :            plane


          2
*** G    V

1  :     -row -plane
2  :          -plane
3  :     -row
4  :  0
```

```
5  :     -row +plane
6  :           plane
7  : -1  -row
8  : -1
9  : +1  -row
10 :  1


          3
*** G    V

1  : -1       -plane
2  : -1
3  :          -plane
4  :  0
5  : +1       -plane
6  :  1
7  :     -row -plane
8  :     -row
9  :     +row -plane
10 :     +row
```

At this moment no storage scheme for the coupled method (velocities and pressures in one matrix), coupling of transport unknowns with velocities, or a complete decoupled method (separate matrices for each velocity component) have been designed. The reason is that at this moment only the pressure-correction method will be implemented. However, in the future other algorithms may be supplied and the storage of these alternative matrices may become important.

16.7 The right-hand side vectors

Array RHSIDE contains the right-hand-side vector for one equation and one time-level. This array is overwritten in each new step of the algorithm.

With respect to the permanent or temporary storage of the right-hand side, exactly the same strategy as for the matrices is used. For that reason ISRHSIDE has dimension NBLOCKS.

Array RHSIDE has the same structure as the local arrays SOLUT, except that no virtual points are used. So in 2D:

RHSIDE(i, j, k) with i = 1 (1) ni+1
j = 1 (1) nj+1
k = 1 (1) NDEG

i,j position of unknown in the grid

k sequence number of unknown in the equation. For a momentum equation k may be 1 to NDIM, for all other equations k=1

16.8 The matrix information arrays INFMM, INFMG, INFMT

Integer information arrays of size 10+nconct*ndim*n, each with the same structure. These arrays are stored temporarily in the integer buffer array IBUFFR for each single block step. That means that after a single block step the arrays are destroyed.

INFMM integer information for Momentum matrix

INFMG integer information for Pressure-gradient matrix

INFMT integer information for Pressure and Transport matrix

PART 1 : positions 1 to 10

pos.1 : Parameter iperiod as stored in IINPUT(56)

pos.2 : ni

pos.3 : nj

pos.4 : nk

pos.5 : ndim

pos.6 : nconct

pos.7 : number of unknowns

pos.8 : Type number indicating the type of matrix
Possible values:

1. momentum
2. pressure gradient
3. pressure
4. transport
5. coupled momentum and pressure equation

pos.9 : -

pos.10 : n, i.e. number of different matrix structures that has been stored.

INFMM(10+i), i= 1, 2, 3, is starting address of storage.
At present the following sequence is used:

1. the matrix code table mct as described in 11.2.5
2. the inverse of the matrix code table invmct as described in 11.2.5
3. the matrix shift table mst for internal rows of the matrix as described in 11.2.5

16.9 The input arrays IINPUT and RINPUT

The reference with respect to the integer buffer array IBUFFR of the integer input arrays is stored in an integer array ISIINPUT of size NBLOCK. This array is available at the host. At the nodes the same type of array is available, however, if block IBLOCK is not available at the node ISIINPUT(IBLOCK) has the value 0.

Array IINPUT contains the integer user input for one block.

The real user input is stored in the arrays RINPUT in exactly the same way. The reference to these arrays is stored in an integer array ISRINPUT of size NBLOCK.

The contents of array IINPUT are defined as follows:

Positions 1-25 contain starting addresses of some integer sub arrays of array IINPUT.
Positions 26-50 contain starting addresses of some real sub arrays of array RINPUT with respect to RINPUT.
Positions 51-100 contain some general constants.
In positions 101-150 length information with respect to the arrays with starting addresses in positions 1-50 are stored.
In the last part the sub arrays of IINPUT are stored, where the starting addresses are defined by the first 25 positions of IINPUT.

Detailed contents:

Pos.	Parameter	meaning
1	IPIINCOF	Starting address of array IINCOF
2	IPIINBC	Starting address of array IINBC
3	IPIINCND	Starting address of array IINCND
4	IPIINTIM	Starting address of array IINTIM
5	IPIINSOL	Starting address of array IINSOL
6	IPIINDSC	Starting address of array IINDSC
7	IPIINBLK	Starting address of array IINBLK
8	IPIINTUR	Starting address of array IINTUR
9	IPIINCOM	Starting address of array IINCOM
10	IPIINSEQ	Starting address of array ISEQEQ
11	IPIINCAV	Starting address of array IINCAV
12	IPIINFREESURF	Starting address of array IINFREESURF
13	IPCLUSTER	Starting address of array ICLUSTER
14	IPIGLOBSTRUC	Starting address of array IGLOBSTRUC
15	IPILOCSTRUC	Starting address of array ILOCSTRUC
16	IPIPROFILE	Starting address of array IPROFILE
17	IPIFREQOUT	Starting address of array IFREQOUT
26	IPRINCOF	Starting address of array RINCOF
27	IPRINBC	Starting address of array RINBC
28	IPRINCND	Starting address of array RINCND
29	IPRINTIM	Starting address of array RINTIM
30	IPRINSOL	Starting address of array RINSOL
31	IPRINTUR	Starting address of array RINTUR
32	IPRINBLK	Starting address of array RINBLK
33	IPRINDSC	Starting address of array RINDSC
34	IPRINCOM	Starting address of array RINCOM
35	IPRINCAV	Starting address of array RINCAV
36	IPRINFREESURF	Starting address of array RINFREESURF
37	IPRGLOBSTRUC	Starting address of array RGLOBSTRUC
38	IPRLOCSTRUC	Starting address of array RLOCSTRUC
39	IPRPROFILE	Starting address of array RPROFILE

Pos.	Parameter	meaning
51	NI	Number of cells in "i"-direction
52	NJ	Number of cells in "j"-direction
53	NK	Number of cells in "k"-direction
54	NVIRTUAL	Number of virtual cells
55	NDEGFD	Number of degrees of freedom: 3+ntrnsp+nturb
56	IPERIOD	Indication of periodical directions along the boundaries. IPERIOD consists of 3 digits, where the digits are numbered 321 Each digit represents one direction. The digits may take the following values: 0: no periodical boundary condition 1: periodical boundary condition 2: anti-symmetric periodical boundary condition
57	NDIM	Dimension of space
58	VLEN	Number of virtual tangential velocities needed from neighbouring blocks
59	IGEO	Type of geometry discretization Possible values: 1: Original ISNaS approach 2: Improved present ISNaS approach
60	MAXCON	Maximal number of connections in a row for all matrices, including the main diagonal
61	MAXB	Maximal number of real values for one boundary condition
62	MAXMAT	Maximum number of different stencils for one matrix
63	NTRNSP	Number of transport equations
64	NTIMLV	Number of time levels in solut
65	MAXDX	Maximum distance of the end points of the molecules to the centroid. This distance is measured in global coordinates in one direction.
66	NCOEFS	Maximum number of different coefficients for one equation
67	POSTTYPE	Type of interpolation for postprocessing. Possible values: 0: Interpolation in physical space 1: Interpolation in computational domain
68	NMACHINES	Number of parallel processors to be used.
69	MAXVPOINTS	MAX (ni+1+2*nvirtual)*(nj+1+2*nvirtual) in 2-D MAX (ni+1+2*nvirtual)*(nj+1+2*nvirtual)*(nk+1+2*nvirtual) in 3-D
70	NTURB	Number of extra transport equations for turbulence
71	MAX_POINTS	Maximum number of points in a single block
72	MAX_CELLS	Maximum number of cells in a single block
73	NBLOCKS	Number of blocks in a multi block method
74	Npoint_global	Global number of nodes
75	MSHTYPE	Indicates type of grid used 0 = staggered grid, 1 = collocated grid

Pos.	Parameter	meaning
76	ICOUPLED	Indicates whether the momentum equations are coupled or not 0 = coupled, 1 = decoupled
77	MCAV	Indicates whether a cavity model is used (1) or not(0).
78	LENGEOM	Help parameter to compute the length of array GEOM
79	LENIGEOM	Help parameter to compute the length of array IGEOM
80	IGEOTIME	If 1 the geometrical quantities depend on time. This is the case if the coordinates of the region depend on time, like in a free su
81	NCLUSTERS	Number of clusters of equations
101	LENIINCOF	Length of first component of array IINCOF
102	LENIINBC	Length of array IINBC
103	LENIINCND	Length of first component of array IINCND
104	LENIINTIM	Length of first component of array IINTIM
105	LENIINSOL	Length of first component of array IINSOL
106	LENIINDSC	Length of first component of array IINDSC
107	LENIINBLK	Length of first component of array IINBLK
108	LENIINTUR	Length of first component of array IINTUR
109	LENIINCOM	Length of first component of array IINCOM
110	LENIINSEQ	Length of first component of array ISEQEQ
111	LENIINCAV	Length of first component of array IINCAV
112	LENIINFREESURF	Length of first component of array IINFREESURF
113	LENICLUST	Length of first component of array ICLUSTER
114	LENIINGLS	Length of first component of array IGLOBSTRUC
115	LENIINLOS	Length of first component of array ILOCSTRUC
116	LENIPROFILE	Length of first component of array IPROFILE
117	LENIFREQOUT	Length of first component of array IFREQOUT
126	LENRINCOF	Length of first component of array RINCOF
127	LENRINBC	Length of array RINBC
128	LENRINCND	Length of first component of array RINCND
129	LENRINTIM	Length of first component of array RINTIM
130	LENRINSOL	Length of first component of array RINSOL
131	LENRINTUR	Length of first component of array RINTUR
132	LENRINBLK	Length of first component of array RINBLK
133	LENRINDSC	Length of first component of array RINDSC
134	LENRINCOM	Length of first component of array RINCOM
135	LENRINCAV	Length of first component of array RINCAV
136	LENRINFREESURF	Length of first component of array RINFREESURF
137	LENRINGLS	Length of first component of array RGLOBSTRUC
138	LENRINLOS	Length of first component of array RLOCSTRUC
139	LENRPROFILE	Length of first component of array RPROFILE

16.10 The coefficients arrays

With respect to the coefficients we are dealing with three different arrays.
Firstly there are two global arrays IINCOF and RINCOF that describe the coefficients. These arrays are part of the global arrays IINPUT and RINPUT, which both have a block dimension. The global arrays IINPUT(iblock) and RINPUT(iblock) are locally copied into local arrays IINPUT and RINPUT. In this way also the arrays IINCOF and RINCOF are copied locally. Besides the arrays that describe the coefficients we have a coefficients array that contains the actual values of the coefficients for one block. The reference with respect to the buffer array of the coefficients

arrays is stored in an integer array ISCOEFS of size NBLOCK. This array is available at the host. At the nodes the same type of array is available, however, if block IBLOCK is not available at the node ISCOEFS(IBLOCK) has the value 0. The coefficients array COEFS is filled before each call of the matrix and right-hand side filling subroutine.

Actual storage:

Double precision array of size NPOINT $*$ NCOEFS in which the coefficients for one differential equations are stored.

COEFS(i,k) contains the k-th coefficient in centroid i. The nodes are ordered in exactly the same way as the scalar equations in SOLUT.

The coefficients array is extended with virtual cells in exactly the same way as the locally copied solution array is.

Array IINCOF consists of a series of one-dimensional arrays of length NCOEFS.
For each equation NCOEFS positions are used, in the case of coupled momentum equations these equations are counted for one equation only. The sequence of the storage in IINCOF is the same as for the unknowns, i.e. first the momentum equations, then the first transport equation etcetera.

Array IINCOF is a two-dimensional array of size NCOEFS $\times$ NTRANS+1.
IINCOF(.,i) refers to equation i, where i=1 is the momentum equation and equations 2, .., NTRANS+1 refer to the transport and turbulence equations in that sequence.
IINCOF(j,.) contains information about the j^{th} coefficient in the following way:
The value of IINCOF(j,.) gives the value of the coefficient in the following way:

let v be the value of IINCOF(j,.) then

v = -1 means that the j^{th} coefficient gets the constant value RINCOF(i)

1<v<100 The j^{th} coefficient is a function of space, time and possibly preceding solutions. The user must provide a user written subroutine USFIL2 (2D) or USFIL3 (3D). The parameter CHOICE in USFIL2 or USFIL3 is identical to v and thus can be distinguished between several possibilities.

Other values of v are not yet allowed.

With respect to the coefficient arrays the following sequence of the coefficients is used:

momentum equations
1. density ρ
2. molecular viscosity μ
3. force_x f_x
4. force_y f_y
5. force_z f_z
6. mass_fraction ε
7. interface_drag β
8. first order coefficient for "x-equation" c_1
9. first order coefficient for "y-equation" c_2
10. first order coefficient for "z-equation" c_3

The coefficients 5 to 10 are only implemented for multi-phase gas flow. The coefficients 8 to 10 are at this moment only implemented for Cartesian grids.

transport equations
1. capacity
2. constant

3. source
4. scalar diffusion or $diff_x x$
5. scalar $diff_x y$
6. scalar $diff_y y$
7. scalar $diff_x z$
8. scalar $diff_y z$
9. scalar $diff_z z$

16.11 Arrays with respect to the boundary conditions

With respect to the boundary conditions we distinguish between 5 types of arrays:

IINBC : contains integer information of the boundary conditions provided by the user

RINBC : contains real information of the boundary conditions provided by the user

INFBND : contains integer information concerning the boundary conditions with respect to the matrix and right-hand side building subroutines

BNDCON : contains real information concerning the boundary conditions with respect to the matrix and right-hand side building subroutines

Let us consider these arrays in more detail:

16.11.1 Array INFBND

Array INFBND is used to indicate the type of boundary conditions for each point at the boundary of each single block for each unknown. In order to make the program flexible we create a separate array INFBND for each equation and each block. Array INFBND is filled in the single block solver. At this moment the pointer array ISINFBND is a more dimensional array of size:

ISINFBND (1:NTRANS+1, 1:NBLOCKS)
Whether this remains is a matter of discussion and investigation.

ISINFBND (i, j) refers to array INFBND corresponding to equation i and relative local block number j.

In 2D we have 4 boundaries of a block:

1. i = 1, j = 1 (1) nj+1
2. i = ni+1, j = 1 (1) nj+1
3. i = 1 (1) ni+1, j = 1
4. i = 1 (1) ni+1, j = nj+1

In 3D we have 6 boundaries of a block:

1. i = 1, j = 1 (1) nj+1, k = 1 (1) nk+1
2. i = ni+1, j = 1 (1) nj+1, k = 1 (1) nk+1
3. i = 1 (1) ni+1, j = 1, k = 1 (1) nk+1
4. i = 1 (1) ni+1, j = nj+1, k = 1 (1) nk+1
5. i = 1 (1) ni+1, j = 1 (1) nj+1, k = 1

6. i = 1 (1) ni+1, j = 1 (1) nj+1, k = nk+1

The momentum equation has equation number 1, the transport equations get sequence number itrans+1, where itrans is the transport equation sequence number.

Each local array INFBND consists of three sub arrays which are stored sequentially:

Part 1: array INFBND_ipoint of length ndim∗2 contains the starting positions of the boundaries in the third part. Hence INFBND_ipoint(1) = 1, INFBND_ipoint(2) contains the starting position of boundary 2 in INFBND_infor and so on.

Part 2: array INFBND_index of length 2∗(ndim-1)∗2∗ndim.
Locally INFBND_index may be considered as a three dimensional array:
INFBND_index(1:2,1:ndim-1,1:2∗ndim).
If we consider INFBND_index(i,j,k) then i=1 denotes the index of the first point at the boundary in the j-direction and i=2 the last point. So in general in R^2 INFBND_index(1,1,k) = 1, INFBND_index(2,1,1) = ni and INFBND_index(2,1,2) = nj.
The index j refers to the tangential directions along the boundaries. In 2D we have only one tangential direction, but in 3D there are two. These tangential directions in 3D are defined as follows:

boundaries 1 and 2: j=1 corresponds to the j-direction, j=2 to the k-direction.

boundaries 3 and 4: j=1 corresponds to the i-direction, j=2 to the k-direction.

boundaries 5 and 6: j=1 corresponds to the i-direction, j=2 to the j-direction.

The index k refers to the boundary sequence number.

Part 3: array INFBND_infor of length sum over all boundary points of all boundaries defines the type of boundary conditions in each point of the boundary. The sequence is first all points at boundary 1, then at boundary 2 and so on. Both vertices and mid side points are filled. As a consequence the vertex points of each rectangle in 2D and the edges in 3D appear at least two times in array INFBND_infor.

The following types of boundary conditions are recognized:

- For the momentum equations:
  1. u_n and u_t given
  2. σ^{nn} and σ^{nt} (in computational domain) given
  3. symmetric periodical boundary condition
  4. anti symmetric periodical boundary condition
  5. symmetric periodical boundary condition, with jump in pressure
  6. anti-symmetric periodical boundary condition, with jump in pressure
  7. rows of cells given (multi block)
  8. vector u given (Cartesian co-ordinates)
  9. a number of quantities given at the boundary (Delft Hydraulics)
  10. u_n and σ^{nt} given
  11. u_t and σ^{nn} given
  12. $|\mathbf{u}|$ and α given
- For the transport equations:
  1. Dirichlet boundary condition

2. Robbins boundary condition
3. Dirichlet boundary condition given at the centroid of the cell (turbulence)
4. Dirichlet boundary condition given at the wall used for wall laws (only heat transfer)
5. symmetric periodical boundary condition
6. anti-symmetric periodical boundary condition
7. rows of cells given (multi block)
8. a number of quantities given at the boundary (Delft Hydraulics)

16.11.2 Array BNDCON

Array BNDCON is used to indicate the value of the boundary conditions for each point at the boundary of each single block for each unknown. In order to make the program flexible we create a separate array BNDCON for each equation, each boundary and each block. Array BNDCON is filled in the single block solver. So the pointer array ISBNDCON is a more dimensional array of size:

ISBNDCON (1:NTRANS+1, 1:NBLOCKS)

ISBNDCON (i, j) refers to array BNDCON corresponding to equation i and relative local block number j, in exactly the same way as INFBND does.

The structure of the local array BNDCON is as follows:

2D momentum equations:

BNDCON(1:max,1:n), where n has the same size as array INFBND_infor. max depends on the type of boundary conditions at the boundary. In general for transport equations max=1, whereas for momentum equations max=ndim. However, in case of boundary conditions of type 9 other values of max may be possible.

BNDCON(i,j) contains the i^{th} boundary condition in point j, where j refers to the same position as in array INFBND_infor. In case of momentum equations the first boundary condition in a point refers to the normal component and the second and third ones to the tangential components.

16.11.3 Array IINBC

Array IINBC is used to indicate the type of boundary conditions for each segment at the boundary of each single block for each unknown. IINBC is defined as sub array of array IINPUT, which means that for each block there is a separate array IINBC.

Contents:

Pos . 1 to ndim∗2: starting address of boundary i of the block with respect to array IINBC. So IINBC(1) = 2∗ndim+1 Boundary is defined in exactly the same way as for array INFBND (same sequence numbers)

For each boundary from the starting position indicated before:

Pos. 1: number of segments the boundary is subdivided into, followed by information of the segments.
In 2D this information is stored in the sequence: i1,i2, itype(1:ntrans+1,1:3) for segment 1, then for segment 2 etcetera. i1 is the initial index of the segment and i2 the end index.
itype(1,1) indicates the type of boundary condition for the momentum equation and itype(i+1,1) for the i-th transport equation.
itype(i,2:3) contains information of how the boundary conditions must be constructed. itype(.,.) is

stored in the mathematical sequence, i.e. itype(1,1), itype(1,2), itype(1,3), itype(2,1), ... and not in the standard FORTRAN sequence. Hence itype may not be considered as a two-dimensional FORTRAN array.
In 3D this information is stored in the sequence:
i1,j1,i2,j2, itype(1:ntrans+1,1:4) for segment 1, then for segment 2 etcetera. i1,j1 is the initial index of the segment and i2,j2 the end index.

The following values of itype(1,1) are possible:

4 Normal component of the velocity vector given, tangential velocity given (Dirichlet type)

5 normal velocity and tangential stress given

6 normal stress and tangential velocity given

7 stress given (Neumann type)

9 law of the wall

10 Cartesian components of the velocity vector given (Dirichlet type)

11 Length of velocity vector vector and angle of inflow given (Dirichlet type)

12 Quadratic velocity profile for the normal velocity, tangential velocity zero (Dirichlet type)

13 A number of quantities given (Delft Hydraulics)

95 anti-symmetric periodical boundary condition with unknown pressure and given jump in flux

96 symmetric periodical boundary condition with unknown pressure and given jump in flux

97 anti-symmetric periodical boundary condition

98 symmetric periodical boundary condition

99 rows of cells given (multi block)

The following values of itype(i,1), i>1 are possible:

1 Dirichlet boundary condition, i.e. T given, where T is the unknown

2 Robbins boundary condition, i.e. aT+(k ∇T)·**n** given

3 Dirichlet boundary condition for turbulence. This type does not have to be given explicitly. Once a wall function is used for the velocity automatically the corresponding boundary conditions for the turbulent quantities are computed.

4 Symmetrical periodical boundary conditions

5 Anti-symmetrical periodical boundary conditions

6 Multi block boundary conditions

7 A number of quantities given (Delft Hydraulics)

27 Dirichlet condition for specific dissipation rate ω.
This type does not have to be given explicitly.

34 Dirichlet condition for temperature at the wall (used for the law of the wall).

48 A special condition for dissipation rate ε (meant for low-Reynolds-number modeling).

itype(i,2:ndim+1) define how the boundary conditions must be computed. In fact the contents strongly depend on itype(i,1). The next table shows which parameters refer to itype(i,2:ndim+1) as function of itype(i,1):

itype(1,1):

1. itype(1,1+j) refers to the j-th velocity component (j=1(1)ndim)
2. itype(1,1+j) refers to the j-th stress component (j=1(1)ndim)
3. itype(1,2) refers to the normal velocity and itype(1,3:ndim+1) to the tangential stress
4. itype(1,2:ndim) refers to the tangential velocity and itype(1,ndim+1) to the normal stress
5. -
6. -
7. itype(1,2) defines the given flux
8. itype(1,2) defines the given flux
9. itype(1,2) defines the type of wall function:

   1 Wall is smooth

   2 Wall is rough, the roughness is stored in rinbc
10. -
11. itype(1,2) defines the normal velocity component
12. itype(1,2) defines the maximal velocity of the profile including the direction (positive = direction of outward normal)
13. -

itype(i,1), i>1:

1. itype(i,2) refers to the value of T
2. itype(i,2) refers to the value of the constant a and itype(i,3) to the right-hand side of the Robbins boundary condition
3. -
4. -
5. -
6. -
7. -
8. -
9. itype(i,2) refers to the value of temperature at the wall

Possible values of itype(i,2:ndim+1) (ivalue)

-1 The parameter is constant, its value is stored in rinbc((iseg-1)∗(ntrans+1)∗ndim+ndim∗(i-1)+j), where j corresponds to itype(i,j+1) and iseg is the segment sequence number with respect to the complete boundary.

1¡ivalue<100 The parameter is a function of time and space. The function subroutine usfunb (ichois,x,y,z,t) is used to define the value. ichois is identical to ivalue.

16.11.4 Array RINBC

Array RINBC is used to store constants for the computation of boundary conditions for each segment at the boundary of each single block for each unknown. RINBC is defined as sub array of array IINPUT, which means that for each block there is a separate array RINBC. The contents of RINBC are defined by IINBC

16.12 Arrays with respect to the initial condition

With respect to the initial condition there are two arrays available:

IINCND : contains integer information of the initial conditions provided by the user

RINCND : contains real information of the initial conditions provided by the user

16.12.1 Array IINCND

Array IINCND is part of array IINPUT and hence is coupled to one block only.

Contents:

One dimensional array of length NDEGFD. In this array the user must put information of the initial condition of all solution vectors that are stored in array SOLUT.

IINCND(i) refers to the i-th solution array in SOLUT, in the following way:

let v be the value of IINCND(i), then

v = 0 means that the i-th solution array does not need to be initialized.

v = -1 means that the i-th solution array gets the constant value RINCND(i)

v < -100 means that the i-th solution array is a function of the space variables. The user must provide a function subroutine FUNC. The parameter choice in FUNC is equal to | v + 100 | and thus can be used to distinguish between several possibilities.

v > 100 means that the array is read from backing storage. The value of v refers to the array number on backing storage.

0<v<100 means that the complete solution is read from the restart file. The value of iincnd(1) gives the sequence number of the solution in this file.

v = 100 , means that the last solution is read.

16.12.2 Array RINCND

Array RINCND is part of array RINPUT and hence is coupled to one block only.

Contents:

One-dimensional double precision array of length NDEGFD that must be filled by the user. RINCND(i) corresponds to IINCND(i) and must be filled by the constant value required, if IINCND(i)=-1. Positions in RINCND not corresponding to values of IINCND(i)=-1 are not used.

16.13 Arrays with respect to the time integration

With respect to the time integration there are two arrays available:

IINTIM : contains integer information of the time integration process provided by the user

RINTIM : contains real information of the time integration process provided by the user

16.13.1 Array IINTIM

Array IINTIM is part of array IINPUT and hence is coupled to one block only.

Contents:

1 : METHOD (Time-integration method) Possible values:

1. Theta method (fixed time step)
2. Second order implicit Gear method (fixed time step)
3. fractional theta method (time step split in 3 parts)
4. generalized theta method(time step split in NFRAC parts)
5. Heun
6. RK-4

2 : -

3 : -

4 : ITIME (Actual time step)

6 : NFRAC (Number of fractional steps for the generalized theta method. ($2 \leq NFRAC \leq 5$)

7 : TVAR (Choice of time stepping methods) Possible values:

0 : Fixed time step DT

1 : Variable time steps with fixed DT per period (NDT periods)

2 : Variable time step dependent on the quality of the approximation

3 : Variable time stepping according to Turek

8 : NDT (Number of different time steps when TVAR = 1, $1 \leq NDT \leq 10$)

9 : -

10 : ISTATIONARY (indicates if the flow is stationary (0) or not (1))

11 : RESTART (writing restart file overwrite (0) append (1) none (-1))

12 : OUTPUT (output to restart file dump (0) Cartesian (1))

13 : FORCE_DELETION (no (0) yes (1))

14 : OUTSESSION (number of session)

15 : NORM (norm to be used for stopping criterion, 0=L_2, 1=max-norm)

16 : NSUBSTEPS_TURBULENCE (number of substeps the time step is subdivided when solving the turbulence equations)

17 : PRESS_CORR_TIME_ITERATION (0 means no iteration, 1 means iteration)

16.13.2 Array RINTIM

Array RINTIM is part of array RINPUT and hence is coupled to one block only.

Contents:

1 : T (Actual time)

2 : T0 (Starting time)

3 : -

4 : -

5 : THETA (Parameter θ for theta method, only used for storage)

6 : TOUTINIT (Indicates initial time for output of results)

7 : TOUTEND (Indicates end time for output of results)

8 : TOUTSTEP (Indicates frequency for output of results)

9-17 : Not yet in use

18 : RELAXATION_PRESS_CORR

19 : ABSACCURACY with respect to pressure correction

20 : RELACCURACY with respect to pressure correction

21 : ABSACCURACY with respect to momentum equations

22 : RELACCURACY with respect to momentum equations

23 : ABSACCURACY with respect to pressure equations

24 : RELACCURACY with respect to pressure equations

25 : ABSACCURACY with respect to transport equations

26 : RELACCURACY with respect to transport equations

27 : ABSACCURACY with respect to turbulence equations

28 : RELACCURACY with respect to turbulence equations

29 : ABSACCURACY with respect to stationary solution

30 : RELACCURACY with respect to stationary solution

31-40 : THETA(1) - THETA(NFRAC) (Parameter θ for various theta methods)

41-50 : TEND(1) - TEND(NDT) (End time, $NDT > 1$ for TVAR=1)

51-60 : DT(1) - DT(NDT) (Time step, $NDT > 1$ for TVAR=1)

16.14 Arrays with respect to the turbulence modeling

With respect to the turbulence modeling there are four arrays available:

IINTUR contains integer information of the turbulence modeling provided by the user

RINTUR contains real information of the turbulence modeling provided by the user

EDDY contains the turbulent viscosity ν_t.

AVERSTR contains the wall stress at no-slip boundaries during the time interval defined by the time window.

16.14.1 Array IINTUR

Array IINTUR is part of array IINPUT and hence to coupled to one block only.

Contents:

1. = NTURB (Number of turbulence equations)
2. = MTURB (Type of turbulence model used)
   Possible values are:

   0 no turbulence model

   1 standard k-ε model

   2 RNG based k-ε model

   3 Wilcox k-ω model

   4 LES model

   5 extended k-ε model
3. = VISCORR (Indicate whether viscous corrections are used or not)
   Possible values are:

   0 without viscous corrections

   1 with viscous corrections
4. = MANISO (Type of anisotropic eddy-viscosity model used)
   Possible values are:

   0 no anisotropic model, i.e. Boussinesq approximation is used

   1 Speziale model

   2 Rubinstein-Barton model

   3 Nisizima-Yoshizawa model

   4 Myong-Kasagi model
5. = LESMODEL (Type of Large Eddy viscosity model used)
   Possible values are:

   1 Smagorinsky model
6. = NWINDOWMAX, i.e. maximum number of time steps used for the time window.
7. = NTIMWIN, i.e. number of time steps in actual the time window.
8. = LENAVER, i.e. number of points in array AVERSTR.

16.14.2 Array RINTUR

Array RINTUR is part of array RINPUT and hence is coupled to one block only. At this moment RINTUR is filled with fixed constants, however, since in practice these turbulence constants may depend on the type of model to be solved, it has been decided to let RINTUR be part of RINPUT. In this way it may be possible in the future to adapt the series of constants in the pre-processing phase.

Contents:

1. $= \kappa$
2. $= E$
3. $= c_\mu$
4. $= \sigma_k$
5. $= \sigma_\varepsilon$
6. $= c_{1\varepsilon}$
7. $= c_{2\varepsilon}$
8. $= c_D$
9. $= \sigma_T$
10. $= A$
11. $= E_{rough}$
12. $= \eta_0$
13. $= \beta_{rng}$
14. $= C_s$ (Smagorinsky constant)
15. $= \alpha$
16. $= \beta^*$
17. $= \beta$
18. $= \sigma$
19. $= \sigma^*$
20. = mod_prod
21. $= c_{3\epsilon}$
22. $= c_{\tau 1}$
23. $= c_{\tau 2}$
24. $= c_{\tau 3}$
25. = time_interval, i.e. the time interval used for time-averaging.

16.14.3 Array EDDY

The turbulent viscosity (eddy viscosity) ν_t, is not only necessary to compute the viscosity coefficient in the momentum equations, but also to compute coefficients in the turbulent transport equations. For that reason this coefficient must be kept during a complete time step.
Since the multi block method may contain loops over the equations separately or over all loops in one, it is necessary to keep the eddy viscosity array for each block.

For that reason EDDY is is referred to by the array ISEDDY of size NBLOCKS. Compare with the co-ordinates. The local array EDDY may be considered as a two-dimensional array containing the values of the eddy viscosity of that block in the centroids of the elements. The storage of such an array EDDY is identical to that of all other coefficients.

16.15 Arrays with respect to the linear solver

With respect to the linear solver there are two arrays available:

IINSOL : contains integer information for the linear solver provided by the user

RINSOL : contains real information for the linear solver provided by the user

16.15.1 Array IINSOL

Array IINSOL is part of array IINPUT and hence is coupled to one block only. For each equation to be solved different information may be required. For that reason IINSOL may be considered as a 25 times (2+NTRANS) array, where IINSOL(.,j) corresponds to the j-th equation. (j=1, momentum equation, j=2, pressure equation, j>2: $(j-2)^{th}$ transport equation.

Contents of array IINSOL(i,j):

i=1 : Contains the length of the array of virtual matrix elements.

i=2 : Type of preconditioner or postconditioner required. Possible values are:

- **0** : No preconditioning nor postconditioning.
- **1** : The matrix is pre-multiplied by a diagonal matrix such that all diagonal elements become 1 (i.e. diagonal scaling).
- **2** : Preconditioning based on an incomplete decomposition of the matrix. Only the diagonal of the matrix is adapted
- **3** : Preconditioning based on an incomplete decomposition of the matrix. The complete non-zero part of the matrix is adapted
- **4** : Preconditioning based on an incomplete decomposition of the matrix. Fill in is allowed.
- **5** : Preconditioning based on Multi-grid
- **100** : Default preconditioning or postconditioning is used depending on the type of equation.
- **-1 to -5** : see 1 to 5, however, now with respect to the postconditioner

i=3 : Control parameter indicating the amount of output required. Possible values:

- **<0** : No output.
- **0** : Only fatal errors will be printed.
- **1** : Additional information about the iteration is printed.
- **2** : Gives a maximal amount of output concerning the iteration process.

i=4 : This parameter controls the start of the iteration process. The following values are allowed:

- **0** : The process starts with as starting vector the null-vector.
- **1** : The process starts with an explicitly given starting vector i.e. the solution is stored in the present time-level In the case of a pressure equation the solver uses alpha (pnew-pold) as starting value, with alpha chosen by the solver
- **2** : The process starts with the solution at the preceding time level, i.e. the solution is stored in the next time-level. In the case of the pressure equation this means: start with the preceding pressure correction
- **3** : The process starts with a linear combination of the solution at two preceding levels (extrapolation)

i=5 : Indicates whether the process must stop on the occurrence of an essential error during the iteration process (0), or control is submitted to the calling program (1).

i=6 : Maximal number of iterations to be performed in each of the solution methods.

i=7 : Parameter indicating the iterative method to be used. Possible values:

- **0** : LSQR
- **1** : CGS
- **2** : GMRES
- **3** : GCR
- **4** : CGSTAB
- **5** :
- **6** : approximate solution using the preconditioner: $\mathbf{x} = \mathbf{P^{-1}f}$
  In the case of an ILU preconditioner, this becomes ($\mathbf{x} = \mathbf{U^{-1}L^{-1}f}$)

i=8 : Parameter indicating the file number to which output should be written.

i=9 : Maximum level of coarse grids (level 0 is finest).

i=10 : Number of times smoother is applied (all levels).

i=11 : Input parameter. After iinsol(11,j) steps the GMRES, GMRESR or GCR procedure is restarted.

i=12 : Input parameter. In the GMRESR or GCR process the residual is made orthogonal to iinsol(12,j) vectors.

i=13 : Input parameter. The subroutine GMRES uses a polynomial preconditioning. The degree of the polynomial is less than iinsol(13,j). Iinsol(13,j) should be less than 100.

i=14 : Number of Ritz values to be computed

i=15 : SAVEPREC, indicates if the preconditioner must be saved or not.

- **0** : build preconditioner each time the equations for a subdomain are solved.
- **1** : build subdomain preconditioner only once when the matrices are built. This is efficient but requires more memory with multi-block problems.

i=16 : PRECBLT, indicates if the preconditioner has been built or not.

- **0** : The preconditioner has not been built.
- **1** : The preconditioner has been built.

i=17 : Indicates if the solution must be printed for each step indicated by ITIME1, ITIME2 and ITSTEP stored in IINSOL(18-20). If IINSOL(17)=0, no solution is printed, if IINSOL(17)=1, the solution is printed.
This possibility is only meant for debug purposes; the solution in contravariant velocity components at the staggered points is printed and no transformation is performed.

i=18 : First time step where printing of the solution should be performed.

i=19 : Last time step where printing of the solution should be performed.

i=20 : Increment of time-steps with respect to the printing. of the solution should be performed.

i=21 : Indication if the matrix to be inverted is singular due to boundary conditions (1) or not (0). The possibility 1 is only available for the pressure matrix.

i=22 : Contains nfill, which is equal to the number of diagonals allowed to become nonzero in the lower (or upper) triangular matrix.

i=23 : If 1 the Neumayer approach is used in CGS, if 0 it is not used.

i=24 : Number of GMRES steps in BCGSTAB(l)

i=25 : If 0 the momentum equations are coupled, if 1 they are not.

Remark: the information with respect to multigrid has been removed. Depending on the requirements given by Jos van Kan multigrid information may be added to this array or a separate array for multigrid.

16.15.2 Array RINSOL

Array RINSOL is part of array RINPUT and hence is coupled to one block only. It has the same structure as array IINSOL.

Contents of array RINSOL(i,j):

i=1 : The iteration is stopped if the residual is less than rinsol(1,j)

i=2 : The iteration stops if the residual is reduced with a factor rinsol(2,j) with respect to the initial residual

i=3 : positions 3 and 4 have been removed from the old description This information is in fact only necessary for the analyze tools of the matrix (computation of eigenvalues of matrices and so-on) Due to the decision to separate solve tools from analyze tools there is no need to store these quantities any more.

i=4 : -

i=5 : Required relative accuracy of the solution only used in GMRES.

i=6 : Required accuracy of the pressure with respect to the divergence-freedom

i=7 : Parameter alpha for the modified ILU preconditioning

i=8 : Relaxation parameter defining the minimum value of omega in the Bi_CGstab process.

i=9-25 : not yet in use

16.16 Arrays with respect to the discretization

With respect to the discretization, all information is stored in array IINDSC, which is part of array IINPUT and hence defined per block. With respect to the type of the discretization there are two arrays available, which are part of array IINPUT and hence defined per block.

IINDSC : contains integer information of the type of discretization provided by the user

RINDSC : contains real information of the type of discretization provided by the user

16.16.1 Array IINDSC

IINDSC may be considered as a two-dimensional array of size 10 x (NTRANS+1): IINDSC(i,j), where j refers to the equation to be solved. j=1 refers to the momentum equation, j>1 to the (j-1)-th transport equation including the turbulence equations.

IINDSC(i,j) is filled as follows:

i=1 : ITERM, indicates which terms are taken into account. Possible values:

0 : the equation is skipped completely, the corresponding solution array is filled in some other way.

1 : all terms of the equation are taken into account

2 : all terms of the equation are taken into account, except the convective terms

i=2 : IDISCRET_METHOD, indicates the type of discretization used.
Possible values:

1 : the classical ISNaS discretization

2 : the Wesseling and van Beek discretization

3 : it is assumed that the viscosity is constant and the flow incompressible. The stress term is reduced since the cross derivatives are eliminated using the incompressibility condition.

4 : the discretization is based on a bilinear interpolation to compute the coefficients of derivatives.

i=3 : indicates if the stress or diffusion terms must be taken into account (0) or not (1)

i=4 : IDISCRET_PRESSURE, indicates the type of discretization used for the pressure (momentum equations only) Possible values:

1 : the classical ISNaS discretization

2 : the Bernard and Kapitza discretization

3 : the Wesseling and van Beek discretization (variant a)

4 : the Wesseling and van Beek discretization (variant b)

i=5 : IUPWIND, indicates the type of upwinding used. Possible values:

0 : no upwinding, i.e. central differences

1 : standard first order upwinding

2 : hybrid central/upwind scheme

3 : higher order upwind scheme (κ-scheme)

4 : TVD scheme with Sweby's Φ-limiter

5 : TVD scheme with R-*kappa* limiter

6 : TVD scheme with MR-*kappa* limiter

7 : TVD scheme with symmetric rational limiter

8 : TVD scheme with PL-*kappa* limiter

9 : TVD scheme with MPL1-*kappa* limiter

10 : TVD scheme with MPL2-*kappa* limiter

11 : TVD scheme with symmetric PL-*kappa* limiter

12 : old first order upwind approach (only for tudfinvol; see Mathematical Manual of tudfinvol)

13 : streamline upwind, method 1 (only for tudfinvol; obsolete now)

14 : streamline upwind, method 2 (only for tudfinvol; obsolete now)

15 : no upwind, i.e. central differences using the old approach (only for tudfinvol; see Mathematical Manual of tudfinvol; obsolete now)

16 : first order upwinding using an old approach (only for tudfinvol; see Mathematical Manual of tudfinvol; obsolete now)

i=6 : IMONOTON_DIFFUSION, indicates the type of scheme used for the discretization of the diffusion term in relation to monotonous schemes. Possible values:

0 : standard scheme (monotony not guaranteed)

1 : half monotonous scheme

2 : monotonous scheme

i=7 : ICOUPLED, indicates whether the momentum equations must be solved in a coupled way (0) or decoupled (1).

i=8 : ILINEAR_CONVECTION, indicates the linearization method used for the convection term in the momentum equations. Possible values:

1 : Newton linearization

2 : Picard linearization

i=9 : ICONSERVATIVE, indicates that a conservative discretization of the convective terms is used (1) or not (0).

i=10 : not yet in use

i=11 : IPRINMT, indicates if the matrix must be printed in the time steps ITIME1 to ITIME2 stored in IINDSC(13-14). Possible values:

0 : The matrix is not printed

1 : The standard matrix is printed. In case of a velocity unknown it is the momentum matrix otherwise it is the transport matrix.

2 : The gradient matrix is printed. This option makes only sense for the momentum equations.

3 : Both the gradient and the momentum matrix are printed. This option makes only sense for the momentum equations.

4 : The pressure matrix is printed. This option makes only sense for the momentum equations.

5 : Both the pressure and the momentum matrix are printed. This option makes only sense for the momentum equations.

6 : Both the pressure and the gradient matrix are printed. This option makes only sense for the momentum equations.

7 : The pressure, momentum and the gradient matrix are printed. This option makes only sense for the momentum equations.

i=12 : IPRINRHS, indicates if the right-hand side must be printed in the time steps ITIME1 to ITIME2 stored in IINDSC(13-14). Possible values:

0 : The right-hand side is not printed

1 : The standard right-hand side is printed. In case of a velocity unknown it is the momentum right-hand side otherwise it is the transport right-hand side.

2 : The right-hand side vector with respect to the pressure is printed. This option makes only sense for the momentum equations.

3 : The right-hand side vector with respect to the pressure as well as for the momentum equations is printed. This option makes only sense for the momentum equations.

i=13 : ITIME1, first time step where printing of matrices and vectors should be performed.

i=14 : ITIME2, last time step where printing of matrices and vectors should be performed.

i=15 : ITIMEST, increment of time-steps with respect to the printing.

i=16 : IPRESGRAD, indicates method to compute pressure gradient (only for tudfinvol). Possible values:

0 : (default) Path integral method using a six point stencil

1 : Path integral method using a three point stencil

2 : Method using auxiliary points between stencil points

3 : Method using four distinct quadrants

4 : Contour integral approach

i=17 : CONTR_VOL, indicates which control volume is used for the momentum equation (only for tudfinvol). Possible values:

0 : (default) Integration over the two triangles adjacent to the considered face.

1 : Integration over the half of the two triangles adjacent to the considered face.

i=18 : ITIMEDERIV, indicates whether the time-derivative is present (0) or not (1).

i=19 : IPRES, indicates whether pressure gradient in momentum equation is present (0) or not (1).

i=20 : CONSISTMOM, indicates whether a consistency investigation is done (1) or not (0).

16.16.2 Array RINDSC

Array RINDSC is part of array RINPUT and hence is coupled to one block only. It has the same structure as array IINDSC. Contents of the array RINDSC(i,j):

i=1 : A parameter κ which specifies the spatial accuracy of the higher order upwind scheme. The value of this parameter must lie in the range [-1,1].

i=2 : A parameter Φ which specifies a member of the family of Sweby's Φ-limiters. The value of this parameter must lie in the range [1,2].

i=3 : A parameter M which has two meanings. First, it is a upper bound used for piecewise linear κ-based limiters. For the non-symmetric limiters, we must have $M \geq 1$. For the symmetric case, the value of this parameter must lie in the range [1,2]. Second, M is used as parameter which specifies a member of the family of symmetric rational limiters. It must lie in the range $[1, \frac{3}{2}]$.

i=4 : A parameter α used for PL_κ, MPL1-κ and MPL2-κ limiters, which its value must lie in the range [-1,0]. Exception must be made for the MPL2-κ limiter, namely $\alpha \in [-1, -\kappa] \cap [-1, 0]$.

i=5-10 : not yet in use.

16.17 Output arrays

The output array contains the solution translated back into Cartesian co-ordinates. If possible information of boundary conditions is stored in this array.
How the output array is structured depends also on the post-processor used. One of the items is the storage for a multiblock method. Is output required for each block separately or for the conglomerate of multi blocks.
The answer(s) to this question is postponed to a later moment.

16.18 Arrays with respect to the block administration and block algorithm

With respect to the block administration we use two separate arrays: array ITOPOL (16.18.2), which contains information of the topology and array IADMIN(16.18.1) which contains information about which blocks on which computer are stored.
With respect to the multiblock algorithm we use two arrays IINBLK (16.18.4) and RINBLK (16.18.5) which are part of the arrays IINPUT and RINPUT. Both arrays contain user information. They are only defined in the global program.

Besides that an array SMSG (16.18.3) is used as temporary array by the multiblock process to store the virtual unknowns for a certain sub face.

An important constant to be used in the multiblock process is the variable ITRANS (16.18.7), which is used to indicate the sequence number of the variable to be transported.

16.18.1 Array IADMIN

Array IADMIN contains information about the blocks stored on the various computers in use.
IADMIN is one dimensional array of size NBLOCKS, which contains for each block the sequence number of the computer on which it is stored. We assume that all the computers on which the single block computations are carried out have sequence numbers 1 to NMACHINES.
So IADMIN(iblock) gives the machine number of single block iblock.

16.18.2 Array ITOPOL

The itopol array stores the connections between the blocks. The basic idea behind the itopol data-structure is that the blocks communicate through common sub-faces. For instance in 2-D, each block consists of four faces numbered 1–4. The faces are divided into sub faces that are numbered from west to east and from south to north, see Fig. 16.18.8. The itopol array stores for each sub face the neighbouring block (0 if the sub face is external) and the corresponding neighbouring face and subface numbers. This means that:

- a sub face of a block is either fully external or fully internal
- an internal sub face of a block is also a sub face of another block (it may also be the sub face of another face of the same block in some special cases with periodic boundary conditions)

Since a sub face that is shared between two blocks may have different directions in the two blocks we must also store the relative orientation of the sub faces. In 2-D, this amounts to a single integer iorient which is 1 if directions are the same and -1 otherwise. In 3-D, this is still a topic of research. The only additional information that is needed to pass information between blocks is the relative orientation of the sub faces. With this information the 'internal boundary' conditions can be passed from block to block.

A sub face in 2-D is described by the index range $i \in [i1, i2]$ where i denotes the number of a grid cell *not* the number of grid points. Because in a general 3-D case the sub faces may have complex shapes which are hard to represent, we restrict the shape of the sub faces:

In 3-D sub faces consist of grid cells in a rectangular index range $[i1, i2] \times [j1, j2]$

In 3-D, the itopol array possibly needs some extensions because sharing of sub faces is not enough to completely determine the coupling between blocks.

The itopol array consists of three parts:

Part I 16.18.2] contains information about parts II and III

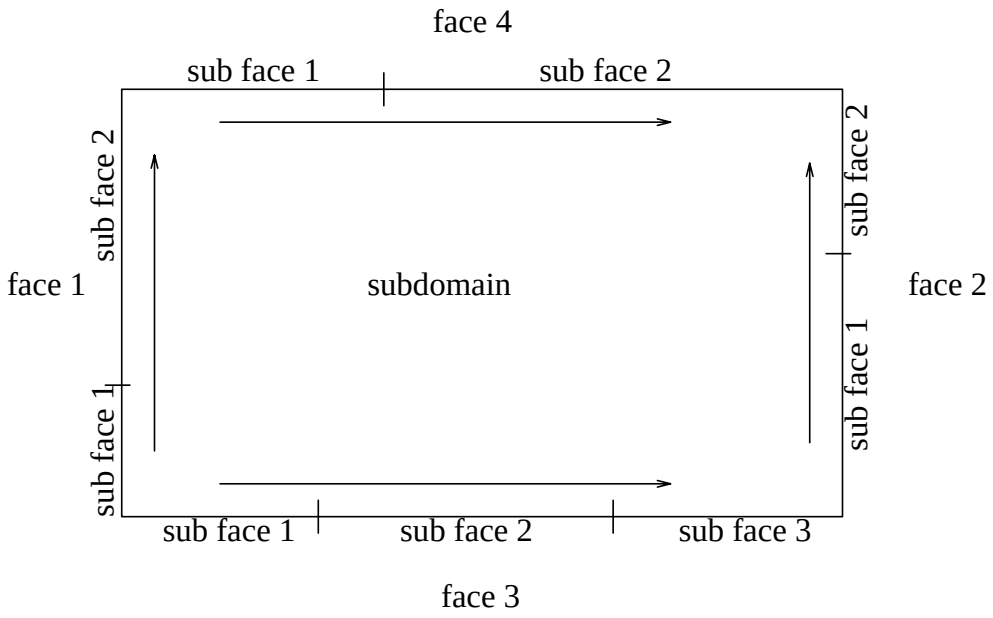

Figure 16.18.8: Directions and numbering of faces and sub faces

Part II 16.18.2] contains the NSUBFACES array.

Part III 16.18.2] contains the SUBFACES array.

Part I:

Pos. 1 nblocks

Pos. 2 nfaces (= 2*ndim)

Pos. 3 maxnsubfaces (maximum number of subfaces encountered on all faces)

Pos. 4 lensubface (current number of integers stored for each subface in the subfaces array)

Pos. 5 Start address of Part II 16.18.2 of this array

Pos. 6 Start address of Part III 16.18.2 of this array

Part II:

Part II is of length nblocks*nfaces and stores the nsubfaces array:

```
integer nsubfaces(1:nblocks, 1:nfaces)
```

The value nsubfaces(i,j) is the number of subfaces at face j of block i.

Part III:

Part III stores the actual multi-block topology in the subfaces array:

```
integer subfaces(1:nblocks, 1:nfaces, 1:maxnsubfaces, 1:lensubface)
```

The value subfaces(i,j,k,m) is the value of position m of the subface record for sub face k of face j of block i. The subface record is organized as follows:

Pos. 1 i1 (lower i index of sub face)

Pos. 2 i2 (upper i index of sub face)

Pos. 3 j1 (lower j index of sub face)

Pos. 4 j2 (upper j index of sub face)

Pos. 5 blockno (number of neighbouring block)

Pos. 6 faceno (number of neighbouring face number)

Pos. 7 subfaceno (number of neighbouring sub face number)

Pos. 8 iorient (relative orientation of the blocks)

Pos. 9 pos (ONLY AVAILABLE INSIDE THE PREPROCESSOR: the starting position of the information for this segment in iinbc 16.11.3)

Pos. 10 pos1 (ONLY AVAILABLE INSIDE THE PREPROCESSOR: the starting position of the information for this segment in rinbc 16.11.4)

In 3-D, the iorient field is not yet defined and always 0. In 2-D, the iorient field is computed based on the following three quantities:

tsign Indicates whether or not the subface has opposite direction when seen from the neighbour. $tsign = 0$ if these directions are the same and $tsign = 1$ if these directions are opposite.

nsign indicates whether or not the sign must be changed when copying normal velocities (*not fluxes*) The normal velocity is the contravariant velocity component corresponding to the normal component. It may therefore point both inward and outward.

relblockor indicates the sign change in $\sqrt{q}$ between this block and its neighbour. This quantity is used in combination with *nsign* to determine the sign change when copying normal fluxes: $V = \sqrt{g}U$. $relblockor = 0$ means no sign change, 1 means a sign change.

The *iorient* field is computed by

$$iorient = tsign + 2 \cdot nsign + 4 \cdot relblockor. \tag{16.1}$$

16.18.3 Array SMSG

The array smsg (sub message) is a one dimensional array which stores the virtual unknowns for a certain sub face. The type of the virtual unknowns is determined by the itrans 16.18.7 parameter. The sub message smsg is manipulated by the routines isddput **??**, isddget **??**, isddvput **??**, isddvget **??**. To describe smsg consider as an example the second sub face of the following lower boundary of a block in Fig. 16.18.9.

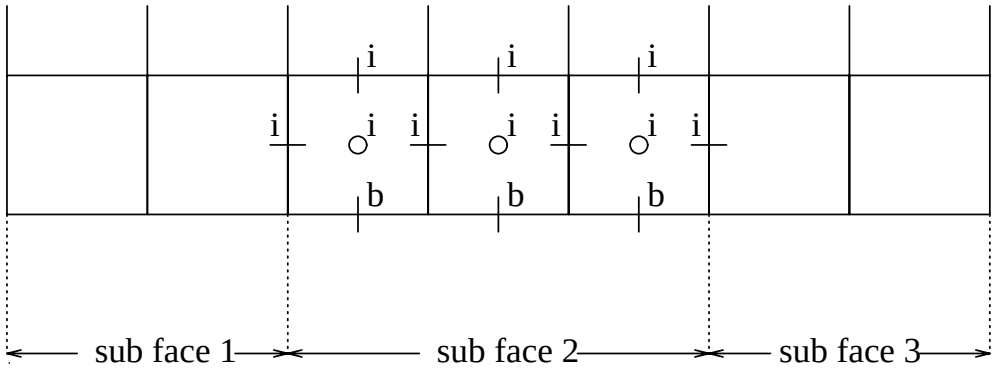

Figure 16.18.9: Unknowns stored in a multi-block sub message

An itrans variable determines the contents of smsg:

$itrans = 1$ all fluxes in Fig. 16.18.9 marked i

$itrans = 2$ all fluxes in Fig. 16.18.9 marked b

$itrans \geq 3$ all pressure/transport unknowns in Fig. 16.18.9 marked i

In the case itrans = 1, the tangential fluxes occur before the normal fluxes. If a subface consists of len grid cells then smsg consists of len unknowns for $itrans > 1$ and 2*len+1 for $itrans = 1$.

Canonical form a sub message

When the orientation of a block is identical to that of its neighbour, the smsg array consists of the unknowns numbered in the increasing coordinate (i, j, k) directions which is consistent consistent with the numbering of grid points of the block the unknowns are coming from. In such cases we have $iorient = 0$ (see itopol Section 16.18.2) and no special precautions (sign changes of unknowns for example) are necessary to transport unknowns from a block to its neighbour.

In a general situation however, the orientations of blocks may be chosen independently of each other. This means that we must reverse the order of copying or change some signs of unknowns when copying from one block to its neighbour. Although the above procedure still gives a unique representation of unknowns for $itrans = 1$ and $itrans > 2$ it is not unique for $itrans = 2$. In the latter case, the unknowns reside on the interface and belong to both blocks and therefore in case of different orientations of sub domains this gives problems.

A solution to this problem is the following. Denote a subface k of side j of block i by (i, j, k). If two blocks share a subface, then we have subface (i, j, k) identical to (i', j', k'). The ambiguity in the representation of a multi-block sub message is now resolved simply be choosing the representation based on the block for which $i \cdot \text{nfaces} \cdot \text{maxnsubfaces} + j \cdot \cdot \text{maxnsubfaces} + k$ is the smallest. This procedure gives a unique representation for all transported quantities $itrans$.

16.18.4 Array IINBLK

Array IINBLK contains information about the multi-block algorithm. IINBLK is an integer array of size (3+NTRANS)∗10 with the following contents:

Pos. 1-10 : General information for all equations

Pos. 11-20 : Information for the momentum equation

Pos. 21-30 : Information for the pressure equation

Pos. 31-40 : Information for the first transport equation

Pos. 41-50 : Information for the second transport equation

⋮

General information:

1: Indication if the sequential or the parallel algorithm is used Possible values:

- **0:** sequential algorithm
- **1:** parallel algorithm

2: ISOL = Indicates the way the subdomains are solved.
Possible values:

- **0:** subdomains are solved accurately (enough). Parallel execution is possible. The domain decomposition process solves a system of interface equations.

1: subdomains are solved inaccurately using the subdomain solver. The GCR-based acceleration technique solves for all unknowns instead of only the interface unknowns. Parallel execution is not possible.

2: the subdomain solution is approximated using the ILU factorization. The same restriction as with ISOL = 1 apply here too.

3: SAVEPREC = indicates if the preconditioner must be saved or not.

0: build preconditioner each time the equations for a subdomain are solved.

1: build subdomain preconditioner only once when the matrices are built. This is efficient but requires more memory with multi-block problems.

4-10: Not yet used (must be 0)

Information with respect to the various equations. Per equation:

1: Indication whether the accelerated or the standard algorithm is used Possible values:

0: accelerated algorithm

1: standard algorithm

2: Maximum number of iterations

3: Dimension of Krylov space in outer loop of GMRESR

4: Dimension of Krylov space in inner loop of GMRESR

5: Control parameter indicating the amount of output required. Possible values:

< 0 : No output.

0 : Only fatal errors will be printed.

1 : Additional information about the iteration is printed.

2 : Gives a maximal amount of output concerning the iteration process.

6: Indicates the truncation strategy to be used. Possible values:

0 : Restart, with an optimized implementation [3], is used instead of truncation

1 : Jackson and Robinson truncation strategy [1]

7: Number of search directions that must be stored

8: Indicates if old values must be reused (1) or not (0) Reusing is only allowed in case of a constant matrix. It makes only sense in combination with IINBLK(7)>0.

9: iortho, defines the kind of orthogonalization used in the GCR method applied to the multiblock process.
Possible values:

0: modified Gram-Schmidt

1: classical Gram-Schmidt with reorthogonalization

10: Not yet used (must be 0)

16.18.5 Array RINBLK

Array RINBLK contains information about the multi-block algorithm. RINBLK is a real array of size (2+NTRANS)∗10 with the following contents:

Pos. 1-10: Information for the momentum equation

Pos. 11-20: Information for the pressure equation

Pos. 21-30: Information for the first transport equation

Pos. 31-40: Information for the second transport equation

⋮

Information with respect to the various equations. Per equation:

1: Relative accuracy of the multiblock algorithm

2: Relaxation parameter for the standard multiblock algorithm

3-10: Not yet used (must be 0)

16.18.6 Array IACTION

The array iaction contains information about the actions to be performed by the host and node programs. The host decides what actions must be performed, e.g. solve an equation, compute velocity correction, print output etcetera. This information is sent to the the node program using routine isacth 4.4.7. The actions are then performed by routine isnmainh 4.2.1 on both the host and the node program. This mechanism of first filling the iaction array with actions and then sending it to the node, replaces the procedure call between host and node routines in a parallel computation.

The iaction array is an integer array of length 10 with information about the actions to be performed. Position i of iaction contains information about action i. If iaction(i) = 0 then action i is not performed. For actions $i \neq 4$, iaction(i) = 1 means that action i must be performed. iaction(4) = ieq means that equation ieq must be solved. iaction(4) = -ieq means that only the virtual unknowns for equation ieq must be received, the equations are not solved in this case. iaction(9) = ieq means that the systems of equations for equation ieq must be built for all subdomains.
Detailed contents:

Action	Meaning
1	stop
2	pressure correction
3	adapt normal fluxes
4	solve equation
5	send field
6	shift solution
7	fill geometrical quantities
8	fill coefficients
9	build equation
10	for compressible flow only: solve equation of state

16.18.7 Constant ITRANS

Sequence number of the quantity to be transported:

1 fluxes near the boundary (V1 + V2)

2 boundary fluxes at block interfaces

3 pressures

> 3 transport quantity

16.19 Arrays with respect to compressible flow

With respect to compressible flow some extra information is stored in the arrays IINCOM and RINCOM.

IINCOM : contains integer information about compressible flow

RINCOM : contains real information about compressible flow

EDDY : contains the values of ρ and $\rho\mathbf{u}$ in the field.

16.19.1 Array IINCOM

Array IINCOM contains integer information about compressible flow. IINCOM is an integer array of size 100 with the following contents:

1 : MCOM Possible values:

- **0** :incompressible
- **1** :compressible (using the pressure correction equation)
- **2** :multi phase gas flow (compressible)
- **3** :fully compressible (using the compressible equations in a standard way)

2 : MCOMUPW, defines whether the equation for the density ρ is discretized by an upwind method or by central differences.
Possible values:

- **0** :central differences
- **1** :first order upwind

3 : MEQUAT, defines the type of equation.
Possible values:

- **0** :standard
- **1** :multi phase gas flow

4 : NO_ENTHALPY, defines whether the enthalpy equation must be taken into account (0) or not (1). In the last case the enthalpy is equal to 0.

5 : NOSCALE, defines if the equations must be scaled (0) or not (1)

6 : IPRESCOR, defines which pressure correction method is used.

7 : MCOMPUPW_method Defines the type of upwinding to be used if MCOMPUPW=1. Possible values:

- **1** : gradient based

2 : mach based

3 : unconditional

4 : explicit

5 : implicit

8 : implexpl indicates if an implicit (0) or explicit scheme is used (1)

9 : itypeequatstat, defines what kind of equation of state is used
Possible values:

0 : perfect gas

1 : user provided equation of state

10 : itypeenergy defines the type of energy equation to be used.
Possible values:

0 : no energy equation

1 : the primary variable is the enthalpy h

2 : the primary variable is the total enthalpy H

3 : the primary variable is the internal energy e

4 : the primary variable is the density times total enthalpy: ρH

5 : the primary variable is the density times total energy: ρE

11-100: Not yet used (must be 0)

16.19.2 Array RINCOM

Array RINCOM contains real information about compressible flow. RINCOM is a real array of size 100 with the following contents:

1 : Gamma

2 : Mach number

3 : Speed of sound

4 : Alpha

5 : P_V

6 : P_OUT

7 : P_REF

8 : RHO_REF

9 : VEL_REF

10 : p_∞

11 : h_∞

12 : $|\mathbf{m}_\infty|$

13-100: Not yet used (must be 0)

It must be noted that items 10-12 are only filled in case of a profile flow (IINPROFILE(1)=1) and boundary conditions that are constant in place and time.

16.19.3 Array EDDY

In compressible flow the extra array eddy is used. EDDY may be considered as a two-dimensional array of size 4 × nsolut, where nsolut is the number of positions in the solution array necessary to store one component.
Array EDDY is organized as follows:

(i,1) is not yet used. It is reserved to store the turbulent viscosity.

(i,2) contains the value of ρ in point i. ρ is stored in cell centers.

(i,3) contains the value of ρV^1 in point i. Hence rho multiplied by the first component of the flux vector. This quantity is stored in the same points as V^1.

(i,4) contains the value of ρV^2 in point i. Compare with position 3.

16.20 Arrays with respect to the sequence of the arrays

There is one array in the preprocessor, also used in the main program that defines the sequence of the equations. This array is array ISEQEQ. The length of ISEQEQ is equal to the number of equations.
ISEQEQ(i) defines the sequence number of equation i.
If ISEQEQ(i) = 0, equation i must be skipped.
The parameter i refers to the standard sequence of the equations, i.e.

1 momentum equation

2 pressure equation

3 first transport equation

and so on.
In case of a decoupled approach 1 to NDIM refer to the momentum equations and NDIM+1 to the pressure equation and so on.
Mark that in fact the equations are solved in the inverse sequence of ISEQEQ.
For example in the standard compressible case first the enthalpy equation (first transport equation) is solved and then the momentum equation followed by the pressure equation.
In that case ISEQEQ has the following contents:
2, 3, 1, meaning that the momentum equation is solved as second one, the pressure equation as third one and the enthalpy equation as first one.

16.21 Arrays with respect to the cavity model

With respect to the cavity model some extra information is stored in the arrays IINCAV and RINCAV.

IINCAV : contains integer information about the cavity model.

RINCAV : contains real information about the cavity model.

16.21.1 Array IINCAV

Array IINCAV contains integer information about the cavity model. IINCAV is an integer array of size 100 with the following contents:

1: IPRES_GAUSS_SEIDEL. If this parameter is 1, the pressure equation is solved by a non-linear Gauss-Seidel accelerated GMRES method

2: ISTARTUP. If 1 the cavitation calculation is started with a modified equation of state.

3-100: Not yet used (must be 0)

16.21.2 Array RINCAV

Array RINCAV contains real information about the cavity model. RINCAV is a real array of size 100 with the following contents:

1 : cliquid

2 : cvapour

3 : lowerpresstransit

4 : upperpresstransit

5-100: Not yet used (must be 0)

16.22 Arrays with respect to the free surface

With respect to the free surface treatment some extra information is stored in the arrays IINFREESURF and RINFREESURF.

IINFREESURF : contains integer information about the free surface.

RINFREESURF : contains real information about the free surface.

16.22.1 Array IINFREESURF

Array IINFREESURF contains integer information about the free surface. IINFREESURF is an integer array of size 10 with the following contents:

1: METHOD. Defines the type of method to be used to update the free surface.
Possible values:

0 : no free surface update

1 : update according to the standard method

2-10: Not yet used (must be 0)

16.22.2 Array RINFREESURF

Array RINFREESURF contains real information about the free surface. RINFREESURF is a real array of size 10 with the following contents:

1 : relaxation parameter

2-10: Not yet used (must be 0)

16.23 Arrays with respect to the structure of the program

With respect to the structure of the program some extra information is stored in the arrays ICLUSTER, IGLOBSTRUC, ILOCSTRUC, RGLOBSTRUC and RLOCSTRUC.

ICLUSTER : contains integer information about the clustering of equations.

IGLOBSTRUC : contains integer information about the global structure of the program

ILOCSTRUC : contains integer information about the structure of the program with respect to each cluster separately.

RGLOBSTRUC : contains real information about the global structure of the program

RLOCSTRUC : contains real information about the structure of the program with respect to each cluster separately.

16.23.1 Array ICLUSTER

Array ICLUSTER contains integer information about the clustering of equations. The length of the array depends on its contents.
contents:

1..NCLUSTERS: Starting addresses of information of each cluster.
The first start address is equal to NCLUSTERS+1.

NCLUSTERS+1 ... Position ICLUSTER(i) of ICLUSTER contains the number of equations in cluster i. The next positions contain the corresponding equation numbers (=sequence numbers of the unknowns)

16.23.2 Array IGLOBSTRUC

Array IGLOBSTRUC contains integer information about the global structure of the program. IGLOBSTRUC is a integer array of size 50 with the following contents:

1 : istationary, defines if the problem is stationary or not. Possible values of istationary:

- 0 : the problem is instationary
- 1 : the problem is stationary, but a time-loop is carried out
- 2 : the problem is stationary, no time loop is carried out

2 : freesurface_flow, indicates if a free surface flow must be computed (1) or not (0).

3 : iterate_initial, indicates if an iteration must be performed to get an accurate initial condition (1) or not (0)

4 : miniter_stationary, gives the minimum number of iterations that must be carried out before checking if a steady state has been reached.

5 : maxiter_stationary, gives the maximum number of iterations that may be carried to compute a steady state.

6 : icrit_stationary, not yet defined

7 :miniter_initial, gives the minimum number of iterations that must be carried out before checking if the initial condition is accurate enough.

8 : maxiter_initial, gives the maximum number of iterations that may be carried to compute the initial condition.

9 : icrit_initial, not yet defined

10-25 :Not yet used (must be 0)

26: Parameter that indicates whether a steady state has been reached (1) or not (0)

27: Parameter that indicates if the initial condition has been converged (1) or not (0)

28: Number of global iterations in a time step that has been carried out.

29-50 :Not yet used (must be 0)

16.23.3 Array RGLOBSTRUC

Array RGLOBSTRUC contains real information about the global structure of the program. RGLOBSTRUC is a real array of size 10 with the following contents:

1 : abs_accuracy_stationary, defines the absolute accuracy to be used for the testing of the convergence to steady state.

2 : rel_accuracy_stationary, defines the relative accuracy to be used for the testing of the convergence to steady state.

3 : abs_accuracy_initial, defines the absolute accuracy to be used for the testing of the convergence of the initial condition.

4 : rel_accuracy_initial, defines the relative accuracy to be used for the testing of the convergence of the initial condition.

5-10: Not yet used (must be 0)

16.23.4 Array ILOCSTRUC

Array ILOCSTRUC contains integer information about the structure of the program with respect to each cluster separately. ILOCSTRUC is a integer array of size (1:NCLUSTERS,1:20) with the following contents:

(i,.) : refers to the i^{th} cluster of equations

(i,1) : istationary, defines if the i^{th} cluster is stationary (1) or not (0). If (1) then no time-loop is performed.

(i,2) : miniter, defines if the minimum number of iterations that must be carried out to reach steady state.

(i,3) : maxiter, defines if the maximum number of iterations that may be carried out to converge to steady state.

(i,4) : icrit, defines the type of criterion to check convergence. Possible values:

0 standard method

1 The criterion is based on $||\delta p|| < \varepsilon$

(i,5) : nsubsteps, defines the number of substeps that must be used in the time integration for this cluster. So if nsubsteps > 1, actually a smaller time step is used for this cluster.

(i,6-10) : not yet defined

(i,11) : Parameter that indicates whether a steady state has been reached (1) or not (0)

(i,12) : Number of local iterations that has been carried out for this cluster.

16.23.5 Array RLOCSTRUC

Array RLOCSTRUC contains real information about the structure of the program with respect to each cluster separately. RLOCSTRUC is a real array of size (1:NCLUSTERS,1:10) with the following contents:

(i,.) : refers to the i^{th} cluster of equations

i,1 : abs_accuracy_stationary, defines the absolute accuracy to be used for the testing of the convergence to steady state.

i,2 : rel_accuracy_stationary, defines the relative accuracy to be used for the testing of the convergence to steady state.

i,3-10: Not yet used (must be 0)

16.24 Arrays with respect to flow around profiles

With respect to flow around profiles (e.g. airfoils) some extra information is stored in the arrays IINPROFILE and RINPROFILE.

IINPROFILE : contains integer information about profile flow

RINPROFILE : contains real information about profile flow

16.24.1 Array IINPROFILE

Array IINPROFILE contains integer information about profile flow. IINPROFILE is an integer array of size 10 with the following contents (note that items 2 and 3 are only filled when MPROFILE=1):

1 : MPROFILE Possible values:

- **0** :no profile flow (e.g. channel flow)
- **1** :profile flow

2 : Curve-number of upper surface of the profile (note: the upper surface of the profile must consist of one curve only)

3 : Curve-number of lower surface of the profile (note: the lower surface of the profile must consist of one curve only)

4-10: Not yet used (must be 0)

16.24.2 Array RINPROFILE

Array RINPROFILE contains real information about profile flow. Note that all items are only filled when MPROFILE=1. RINPROFILE is a real array of size 10 with the following contents:

1 : c: length of the airfoil along the x-axis, for example used to scale the x-coordinate of points at the profile.

2 : α: angle of attack (in radians).

3-10: Not yet used (must be 0)

16.25 Arrays with respect to frequent output

With respect to frequent output some extra information is stored in array IFREQOUT.

IFREQOUT : contains integer information with respect to frequent output.

16.25.1 Array IFREQOUT

Array IFREQOUT contains integer information with respect to frequent output. IFREQOUT is an integer array of size 20 with the following contents (for the definitions of items 6 to 14, see the User Manual)

1: IFREQ, frequency for output, see Users Manual.

2: ILIFT, if 1 the lift and drag are printed, otherwise (0) not.

3: MINMACH, if 1 the minimum and maximum Mach number are printed, otherwise (0) not.

4: NUMBSUPVERT, if 1 the number of supersonic vertices are printed, otherwise (0) not.

5: IPOSSONIC, if 1 the positions of the sonic points are printed, otherwise (0) not.

6: DENSITY_RES_ONE, if 1 then the density residual in 1-norm is printed, otherwise (0) not.

7: DENSITY_RES_TWO, if 1 then the density residual in 2-norm is printed, otherwise (0) not.

8: DENSITY_RES_INF, if 1 then the density residual in ∞-norm is printed, otherwise (0) not.

9: MOMENTUM_RES_ONE, if 1 then the momentum residual in 1-norm is printed, otherwise (0) not.

10: MOMENTUM_RES_TWO, if 1 then the momentum residual in 2-norm is printed, otherwise (0) not.

11: MOMENTUM_RES_INF, if 1 then the momentum residual in ∞-norm is printed, otherwise (0) not.

12: ENTHALPY_RES_ONE, if 1 then the enthalpy residual in 1-norm is printed, otherwise (0) not.

13: ENTHALPY_RES_TWO, if 1 then the enthalpy residual in 2-norm is printed, otherwise (0) not.

14: ENTHALPY_RES_INF, if 1 then the enthalpy residual in ∞-norm is printed, otherwise (0) not.

15-20: Not yet used

16.26 Common block CISNAS

The common block CISNAS is the general ISNaS common block. It is identical on all computers where the program runs at one time. The common block contains only general information. Definition:

```
double precision rmach
integer imach, irefs, ierrors, ioutps
COMMON /cisnas/ rmach(100), imach(100), irefs(100), ierrors(100),
                ioutps(100), iconst(100)
```

The arrays are defined as follows:

imach: array with machine-dependent integer constants.

1:	INTLEN	Defines the length of a real in terms of integers: INTLEN = 1, 64 bits computer INTLEN = 2, 32 bits computer
2:	IINFIN	Largest integer number
3:	NREC4	Number of words that can be stored in one record of the error message file
4:	ICARCN	Indicates if the computer uses the carriage control parameter for output to a file (1) or not (0)
5:	MACHIN	Indicates type of computer (see isnas.env)
6:	NUMCHR	Number of characters in a word
7:	LENWOR	Multiplication factor for record length in open statement if given in words
8:	NREC2	Record length of backing storage file
9-100:	not yet defined	

rmach: array with machine-dependent real constants.

1:	EPSMAC	Machine accuracy
2:	SQREPS	Approximation of the square root of the machine accuracy, usually the single precision accuracy
3:	RINFIN	Largest real number
4-99:	not yet defined	
100:	T	Actual time, to be used for error messages

irefs: array with file reference numbers

1:	IREFRE	Reference number for standard input file
2:	IREFWR	Reference number for standard output file
3:	IREFER	Reference number for error message file
4:	IREF2	Reference number for ISNaS backing storage file
5:	IREF10	Reference number for meshoutput file
6:	IREF17	Reference number for menu message file
7:	IREF73	Reference number for sepcomp.inf
8:	IREF74	Reference number for sepcomp.out
9-100:	not yet defined	

ierrors: array with information of the error status

1:	IERROR	Last error message number found
2:	NERROR	Number of error messages found
3:	MAXERR	Maximal number of errors allowed before the program is terminated
4:	IWARN	Last warning number found
5:	NWARN	Number of warnings found
6:	MAXWAR	Maximal number of warnings allowed before printing of warnings is suppressed
7-100:	not yet defined	

ioutps: array with information of output and debug parameters

1:	IOUTPT	Defines level of output Possible values: -1: all output is suppressed, unless explicitly required 0: standard minimal output. With respect to the iteration processes only the end information is printed. 1: extended output. Also information with respect to the iteration processes is printed. 2: extra output. In this case also the files read are echoed. 3: maximal output. See 2, however all actions performed by memory management subroutines are writt
2:	IDEBUG	Debug level
3:	ITIME	Indicates if the CPU time must be printed after each call of a main subroutine (1) or not (0)
4:	IBLOCKRF	Actual block number in multi block process
5:	ITERAT	Actual iteration number in global iteration process
6-100:	not yet defined	

iconst: array with general constants

1:	ILEFT	Internal sequence number of left-hand boundary in computational domain (2D)
2:	IRIGHT	Internal sequence number of right-hand boundary in computational domain (2D)
3:	IDOWN	Internal sequence number of lower boundary in computational domain (2D)
4:	IUP	Internal sequence number of upper boundary in computational domain (2D)
5-100:	not yet defined	

16.27 Common block CUSER

The common block CUSER is a special ISNaS common block to be used by developers only. This common block is meant to pass parameters from the input read in the part USER INPUT to user subroutines or subroutines adapted by users. The purpose of this common block is mainly for debugging.

Definition:

```
double precision user
integer iuser
COMMON /cuser/ user(1000), iuser(1000)
```

The arrays IUSER and USER are read by the input part in the sequence indicated by the user.

Bibliography

1. C.P. Jackson and P.C. Robinson. A numerical study of various algorithms related to the preconditioned conjugate gradient method. *Internal Journal for Numerical Methods in Engineering*, 21:1315–1338, 1985. 16.18.4

2. G. Segal, K. Vuik, W. Kuppen, and M. Zijlema. ISNaS - incompressible flow solver. mathematical manual. Report 93-96, Delft University of Technology, Faculty of Technical Mathematics and Informatics, Delft, The Netherlands, 1993. 11.1, 11.2

3. C. Vuik. Solution of the discretized incompressible Navier-Stokes equations with the GMRES method. *Int. J. Num. Meth. in Fluids*, 16:507–523, 1993. 16.18.4