SEPRAN

SEPRA ANALYSIS

PROGRAMMERS GUIDE

GUUS SEGAL

SEPRAN PROGRAMMERS GUIDE

May 2010

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

Contents

1 Introduction

1.1 General remarks

In this manual it is described how you can create your own programs by SEPRAN. In the INTRODUCTION and the Users Manual it is described how you may create a mesh by SEPMESH, solve a problem by SEPCOMP or perform postprocessing by SEPPOST. It is even indicated how you may extend these programs with function subroutines or your own element subroutines. However, it may be possible that the options provided in the Users Manual are not sufficient for your own purposes and that you want to write your own main program.
In this manual it is described how you may create your own FORTRAN program using all the tools SEPRAN provides. In this way you can solve practically any problem.
This manual contains the following chapters:

Chapter 1 contains some general information concerning SEPRAN,

Chapter 2 treats the general structure of a SEPRAN program. Without reading this chapter it is practically impossible to create your own main program.

Chapter 3 is devoted to mesh generation. All alternatives to SEPMESH are treated as well as the structure of the files to be created.
This chapter is especially useful if you intend to change the mesh during computation for example for free boundary problems.

Chapter 4 treats the starting subroutines as well as the routines reading the problem definition.

Chapter 5 treats all subroutines related to boundary conditions, initial conditions and creation of SEPRAN vectors.

Chapter 6 deals with the filling of coefficients for the standard elements treated in the manual Standard Problems.

Chapter 7 is devoted to the building of the large system of equations.

Chapter 8 treats the linear solvers.

Chapter 9 treats the non-linear solvers.

Chapter 10 introduces the possibility to compute eigenvalues and eigenvectors of physical problems.

Chapter 11 gives a number of possibilities to solve time-dependent problems.

Chapter 12 shows how SEPRAN vectors may be manipulated. For example algebraic manipulation like adding two vectors and so on.

Chapter 13 deals with the computation of boundary and volume integrals of the solution or other SEPRAN arrays.

Chapter 14 treats how derived quantities like derivatives may be computed.

Chapter 15 gives a number of possibilities to provide print output of SEPRAN vectors during computation.

Chapter 16 treats various plot possibilities ranging from plotting of the mesh to plotting of special quantities related to the solution.

Chapter 17 deals with free surface and moving boundary problems.

Chapter 18 gives a number of special mapping subroutines. For example to map a SEPRAN array in user array or vice-versa. But also to map a SEPRAN matrix from one structure into another one.

Chapter 19 treats some auxiliary subroutines which may be of use in your main program.

Chapter 20 shows how you may write results of SEPRAN computations to files, for example for post-processing purposes or to use in future computations.

Chapter 21 gives an overview of some important SEPRAN commons. This chapter is only useful for the very experienced SEPRAN programmer.

Chapter 22 treats the SEPRAN memory management subroutines. This chapter is only useful for experienced SEPRAN programmers who want to change existing SEPRAN subroutines or want to create new ones.

Chapter 23 deals with the creation of error messages in subroutines.

Chapter 24 treats the main SEPRAN arrays that are available to the user. In fact this chapter contains the complete data structure. Of course this chapter is only useful for experienced users.

Chapter 25 gives a number of examples of SEPRAN programs

Chapter 25.5.1 contains the references.

Chapter 27 contains an index of some keywords.

1.2 Some remarks concerning the storage of data

SEPRAN distinguishes between arrays of variable length and arrays of fixed length, see Chapter 2. In fact all data that occupy a considerable amount of storage, are stored either on backing storage or in a large buffer array. The SEPRAN arrays contain information of where and how these data are stored. In general as much as possible data that will be used frequently are kept in-core, including the large matrices. If it is not possible to keep data in-core, SEPRAN puts information on backing storage (file 3) automatically. When the large matrices take more than half the available buffer space, they are splitted and kept out-of-core.
The large buffer array is stored in blank common:

```
COMMON  IBUFFR ( . . . )
```

The length of this array is a machine dependent quantity, that is set at the installation of the package. When the user wants to extend this array he must do the following things:

- Declare a buffer array in blank common in the main program:

  ```
  COMMON IBUFFR ( xxxx )
  ```

  with xxxx large enough. Inform at your local installation personal which is the standard length and which is the maximal length of this array. For most UNIX and MSDOS computers the default length is set equal to 5,000,000. Sometimes, however, only 1,000,000 is used.

- Make clear to SEPRAN what the actual length of the buffer array is. This may be done in several ways, depending on the application.
  For example

  - If program sepmesh is used, but the size of IBUFFR must be enlarged:

    ```
    program sepmesh
    implicit none
    integer nbuffr
    parameter (nbuffr=xxxx)
    integer ibuffr
    common ibuffr ( nbuffr )
    call sepmsh ( nbuffr )
    end
    ```

  - If program sepcomp is used, but the size of IBUFFR must be enlarged:

    ```
    program sepcomp
    implicit none
    integer nbuffr
    parameter (nbuffr=xxxx)
    integer ibuffr
    common ibuffr ( nbuffr )
    call sepcom ( nbuffr )
    end
    ```

– If one of the starting subroutines is used that actually has the possibility to give the length of the buffer array (for example sepstl):

```
program main
implicit none
integer nbuffr
parameter (nbuffr=xxxx)
integer ibuffr
common ibuffr ( nbuffr )
integer kmesh(...), kprob(...), intmat(5)
call sepstl ( kmesh, kprob, intmat, nbuffr )
.
.
.
end
```

– If one of the starting subroutines is used that does not have the possibility to give the length of the buffer array (for example sepstr):

```
program main
implicit none
integer nbuf
parameter (nbuf=xxxx)
integer ibuffr
common ibuffr ( nbuf )
integer nbuffr, kbuffr, intlen, ibfree
common /cbuffr/ nbuffr, kbuffr, intlen, ibfree
integer kmesh(...), kprob(...), intmat(5)

call sepstr ( kmesh, kprob, intmat )
nbuffr = nbuf
.
.
.
end
```

Hence in this case common block cbuffr must be used (See Section 21.2) and nbuffr must get the actual length of array IBUFFR. Since the starting subroutine (for example SEPSTR 4.2.1) initializes all common blocks including CBUFFR, it is necessary to put the statement NBUFFR = NBUF, after the call of the starting subroutine.

1.3 Debug facilities

Besides the usual facilities provided by compilers, etc., SEPRAN offers some extra debug possibilities.
At this moment the following options are available:

1. Debugging the memory management
2. Producing extra print output

Debugging the memory management

In order-to activate the debugging of the memory management the user has to provide a file named `sepran.dbg` in the directory where he is executing the SEPRAN job.

In the file `sepran.dbg` debug commands may be given. For these commands the same rules as for a standard SEPRAN input file may be applied (see Users Manual Section 1.4), including comments. The following commands are recognized:

```
debug_level = d
nbuffr = n
int_infin = i
no_swap
swap_on
```

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: $2^{31} - 1$.

`no_swap` prevents that the SEPRAN swapping mechanism is activated. If not enough space is available in the buffer an error is printed and the program halted. Extra information concerning the contents of the buffer is printed so that the reason of the full buffer can be detected.

This is presently the default.

`swap_on` activates the SEPRAN swapping mechanism.

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.

Producing extra print output

Besides checking the memory management it is always a good practice to produce extra output in case an error occurs that can not be detected immediately or if results can not be trusted. The first option to produce extra output is to use `set output_level=3` in the input file. This produces a lot of extra information of the subroutines called and when memory management is activated.

A different option is to produce extra output by using the input block `debug_parameters` as described in Section 1.4 of the Users Manual.
This input block has the following structure:

```
DEBUG_PARAMETERS
   parameter_1
   parameter_2
      .
      .
END
```

All these parameters must be given at a new line. Each activates a separate output part.
The debug parameters may be either debug followed by an underscore and a subroutine name or a specific print or plot command. The following list of debug_subroutines are available:

```
debug_buildbf
debug_compstat01
debug_erintpol
debug_extfreesubr
debug_fvcvdf
debug_fvincnd
debug_fvpres
debug_fvprmom
debug_fvsbstep
debug_fvstart
debug_fvtstep
debug_fvviscmatiq2
debug_fvviscmatiq2s
debug_hollowmesh
debug_initfree
debug_mshcrt
debug_mshfillkmeshad
debug_mshobsfl
debug_mshobstmarkpt
debug_mshobstmesh
debug_mshsortel3d
debug_mshsrf
debug_mshvlm
```

```
debug_pltrac
debug_prinmtbf
debug_prob01crobst
debug_probdfbf
debug_probflkprobaa
debug_probskippoint
debug_prorea
debug_mshobstpoint
```

The option `debug_name` means that debug statements in subroutine `name` are activated. You have to have knowledge of the SEPRAN sources in order to know which of these options makes sense.

The following list of print and plot debug commands is available:

```
plotedges
plotobstacles
plotskipelements
printobstacles
printskipelements

start_surface
number_of_rotations

write_geomview
write_iinput_mesh
write_matlab_mat
format_matlab =  i
```

These parameters have the following meaning:

plotedges Plots edges with edge numbers (2d only)

plotobstacles Using this options activates the plotting of obstacles, elements in obstacles and nodal points in obstacles. If cross-sections with a fixed mesh are computed, also these intersections are plotted.
In 3D the mesh and the obstacle are plotted in slices.
The starting surface is defined by `start_surface`.

plotskipelements is used to plot all elements that are skipped. This is meant for obstacles in combination with skip elements.

printobstacles activates printing of extra information about obstacles as well as elements and nodes lying in it and possibly intersections with a fixed mesh.

printskipelements is used to print all elements that are skipped. This is meant for obstacles in combination with skip elements.

start_surface Defines the starting surface in 3D for plotting of slices of elements.
Default: 1

number_of_rotations In case of 3D plots each plot is rotated over a complete circle subdivided into *number_of_rotations* parts.
Default: 1

write_geomview Using this keyword means that information of the 3D mesh and the obstacle in it is written to a series of files with name `openview_xxx.off`. `xxx` is a sequence number starting with 001. This file is in open view format and suitable to be plotted with the freeware software package `openview`.

write_matlab_mat This keyword activates the writing of matrices and right-hand-side vectors to specific files in matlab format. This is especially meant for testing in matlab. See subroutines PRCONGMATLAB1, PRCONGMATLAB2 and PRMATLABMAT and such subroutines for further explanation.
Default: not

format_matlab Defines the format used by the matrices written.
If 0 a standard format using row and column index for a matrix element is applied.
If 1 a one-dimensional storage is applied. This is more efficient in matlab, but for large matrices you get integer overflow. So this method is only recommended for small problems. Default: 0

Besides that there are a number of extra options:

```
check_hanging_nodes
checkobstconsistency
checksubtetrahedrons
check_sign_jacobian
trace
```

These parameters have the following meaning:

check_hanging_nodes This option is meant to check the mesh. It detects if there are nodes that are positioned somewhere on an edge but are not part of all elements connected to this edge. Such hanging nodes should be avoided, unless special measures are taken.

checkobstconsistency This option is meant to check the consistency of the arrays ielemobst and ipointsobst with respect to nodes that are completely inside the obstacles, i.e. only lying inside elements that are inside the obstacle.

checksubtetrahedrons This option is meant to check the subtetrahedrons that are created if a 3D mesh is subdivided because of the presence of an obstacle.

check_sign_jacobian Check if the sign of the Jacobian of all elements is positive. If not give an error message.

trace If this parameter is used each time a subroutine EROPEN or ERCLOS is called a print is made with the name of the calling subroutine and the relative level in a list of calling subroutines. EROPEN is called at the beginning of most subroutines and ERCLOS finishes the same subroutine.
Hence with this option it is easier to trace where an unexpected error occurs.

These debug options activate debug parameters in common block CDEBUG_ROUTS as described in Section 21.22.

1.4 Tools

Besides the standard SEPRAN commands `sepcomp, sepmesh, sepfree` and `seppost` and the command `seplink` treated in the Users Manual, SEPRAN has some extra tools that may be used for the development and checking of SEPRAN programs: The following tools are available:

```
seplink
sepdbg
compile
compiledbg
sepget
sepgetall
sepgetex
sepdiff
sepclean
sepexpc
sepfind
sepfindall
get-key
calltree
check
sepcheck
sepvar
change_for
```

The last set of commands is only available if you have the commercial package FORCHECK of Leiden University at your disposal. This package is not part of SEPRAN, nor is it delivered by SEPRA. The commands mentioned above have the following syntax and meaning:

seplink is used to compile and link SEPRAN programs.
Usage: `seplink main_program file1.f file2.f ...`
It expects the existence of the file main_program.f, containing the main program, and the files file1.f file2.f ...
seplink compiles the files main_program.f and file1.f file2.f ... and links them with all $*.o$ files in the directory as well as the SEPRAN library in order to create the executable main_program.

sepdbg is used to compile and link SEPRAN programs with the debug option.
Usage: `sepdbg main_program file1.f file2.f ...`
In fact sepdbg is identical to seplink, however, it compiles all subroutines with the debug option on, making it possible to use the standard unix debugger.

compile compiles a set of subroutines.
Usage: `compile file1.f file2.f ...`
It expects the existence of the files file1.f file2.f ...
compile compiles the files file1.f file2.f ... creating the files file1.o, file2.o ...

compiledbg compiles a set of subroutines with the debug option.
Usage: `compiledbg file1.f file2.f ...`
compiledbg has the same meaning as compile, however, it compiles with the debug option on.

sepget copies standard SEPRAN subroutines into the local directory.
Usage: `sepget subr1 subr2 subr3 ...`
The files subr1.f, subr2.f subr3.f ... if they exist and are part of the SEPRAN directories are copied into your local directory.

sepgetall copies standard SEPRAN subroutines into the local directory.
Usage: `sepgetall subr`

sepgetall has the same meaning as sepget, however, it copies all files that contain the text subr as part of its names.

sepgetex copies all files corresponding to a standard SEPRAN example to your local directory, provided these files have been stored in one of the subdirectories of SEPRAN/sourceexam, where SEPRAN is the main SEPRAN directory.
Usage: `sepgetex example`
for example: sepgetex potential.

sepdiff compares local files with the standard sepran files and stores the differences in files.dif.
Usage: `sepdiff subr1.f subr2.f subr3.f ...`
The files subr1.f, subr2.f subr3.f ... are compared with the standard SEPRAN files subr1.f, subr2.f subr3.f ..., and the differences are stored in the files subr1.dif, subr2.dif, subr3.dif. Like in all unix commands, wild cards are allowed, hence sepdiff $*.f$ compares all local files with SEPRAN files.

sepclean Removes all files in the local directory that are identical to the SEPRAN subroutines.
Usage: `sepclean`

sepexpc Expands FORTRAN sources, containing calls to SEPRAN subroutines.
Usage: `sepexpc file1.f file2.f file3.f ...`
If in the sources of the subroutines calls to SEPRAN subroutines are available in the form

```
call seprout()
```

where seprout is some SEPRAN subroutine, then the source text is updated, and the call is replaced by a call containing the complete parameter list. Of course exactly the names used in the source of the SEPRAN subroutines are used, so the user must adapt the calls if he uses different names.

sepfind Returns with the place where the source of the subroutine is stored.
Usage: `sepfind subr`
where `subr` is a standard SEPRAN subroutine.

sepfindall Returns with the place where the source of all subroutines containing a specific text are stored.
Usage: `sepfindall text`
The position of all subroutines of which the names contain the verb+text+ literally is returned. This can be used for example to find all subroutines that contain a certain text in the name.

get-key Return with the names of all subroutines that contain the specific keywords in their keyword list.
Usage: `get-key keyword1 keyword2 ...`
The names of all subroutines that contain the keywords `keyword1` and `keyword2` and all others mentioned in the list of keywords is returned. With and we mean the logical and operator.

calltree produces a tree of calling SEPRAN subroutines.
Usage: `calltree subr`
which gives the names of all subroutines that are called by `subr` or

```
calltree subr 3
```

which gives the names of all subroutines that are called by `subr`, as well as the names of all subroutines that are called by this subroutines and this three levels deep. 3 may be any positive number but only small values produce a readable result. Finally the number may also be negative like

```
calltree subr -1
```

which gives the names of all subroutines that call `subr`. Higher negative values produce a tree that is several levels deep.

check check FORTRAN sources. This command can only be used if you have FORCHECK at your disposal.
Usage: `check file1.f file2.f ...`
The FORTRAN source of all files file1.f, file2.f ... are checked.

sepcheck checks whether all variables in the source code have been described and if all error numbers have been described. sepcheck can only be used if you have FORCHECK at your disposal. It requires that the FORTRAN source is written according to the standard SEPRAN layout.
Usage: `sepcheck [-cnrv] files`
The various options have the following meaning:

-c The comment block of the files specified are checked on the following points:

- the version may not be raised more than one. If the main number is raised the sub-number must be equal to 0.
- all keywords must appear in the list of allowed keywords
- the order of the descriptions of the arguments must be alphabetic
- all variables must be described
- all described variables must appear
- the description of the called subroutines must be in alphabetic order
- the described subroutines must appear
- all called subroutines must be described
- the description of the error messages must be in ascending order
- all error messages must be described
- the described error messages must appear
- a call to eropen and erclos must have the name of the subroutine as argument
- all comment lines must begin with a lower case `c`

All remarks are written to the file `report.out`

-n Same as -c but now without checking the names of the variables

-r Checks whether all error numbers in all source files of the SEPRAN directories make up the total list of error numbers. The output is written to the file `error.out`.

-v Checks whether the version block of the altered file minus the first line contains at least all that is in the version block of the original file. The output is written to the file `version.out`.

sepvar Makes a list of all variables, subroutines called and error messages provided with a description as stored in the SEPRAN database. This command can only be used if you have FORCHECK at your disposal.
Usage: `sepvar file1.f file2.f file3.f ...`
The output is written in the file `vars.out`

change_for Updates the source files in order to get a more uniform layout.
Usage: `change_for file1.f file2.f file3.f ...`
change_for produces strange effects if the subroutine is not written according to the standard SEPRAN layout, including all comment blocks.

2 General structure of a SEPRAN program

2.1 Introduction

If the user does not want to use the standard program SEPCOMP, he may write his own main program. Reasons to do this are for example that SEPCOMP does not contain the option required, or that the user wants to do some extra computations or perform extra output, which can not be included in SEPCOMP easily.
For that reason SEPRAN allows the user to write his main program, which at the users wishes may also contain pre- and postprocessings parts.
The advantage of this approach is the large flexibility of SEPRAN, but of course if only standard problems must be solved then one better uses one of the standard main programs. The programming language is standard FORTRAN 77.
In this section we shall treat the general structure of SEPRAN programs. This is done by means of a simple computational program that solves a linear problem. Of course this program does not add anything to the main program SEPCOMP, but it is an easy program to show the ideas of SEPRAN. In this program we assume that SEPMESH has been used to generate the mesh, and that SEPPOST will be used to perform the post-processing.

A simple SEPRAN main program may have the following structure:

```
      source code                                              Reference

 program sepmain                                                  1
 implicit none                                                    2
 integer lmesh, lprob                                             3
 parameter (lmesh=1000, lprob=2000)
 integer kmesh(lmesh), kprob(lprob), isol(5), intmat(5)
 data kmesh /lmesh*0/, kprob /lprob*0/, isol /5*0/,               4
+     intmat /5*0/

 kmesh(1) = lmesh                                                 5
 kprob(1) = lprob
 call sepstr ( kmesh, kprob, intmat )                             6
 call presdf ( kmesh, kprob, isol )                               7
 call linsol ( kmesh, kprob, intmat, isol )                       8
 call output ( kmesh, kprob, isol )                               9
 call finish ( 0 )                                               10
 end                                                             11


 If necessary function subroutines to define                     12
 coefficients or boundary conditions.
```

Explanation:

1. PROGRAM statement, standard FORTRAN 77.
   The name of the program is arbitrary.

2. implicit none
   This non-standard FORTRAN 77 statement is recognized by practically all FORTRAN 77 compilers. It means that all variables, arrays etc. should be declared explicitly. If you forget to declare one of the variables an error message will be printed. The use of this option makes it possible to find some of your typing errors in an early stage by the compiler.

3. Declarations, consisting of an integer statement, followed by a parameter statement, followed by the actual declarations.
   The SEPRAN arrays must be declared. We distinguish between arrays of fixed length (INTMAT, ISOL) and arrays of variable length (KMESH, KPROB). In these arrays some integer information is stored. Using the concept of common blocks makes it possible for SEPRAN to give actual information as co-ordinates, coefficients etcetera to each subroutine. The user does not have to know how this process takes place, however, he is able to manipulate the actual information flow through the integer SEPRAN arrays. In fact the integer arrays act as pointers.

   In array KMESH information concerning the mesh is stored. KMESH is a variable length array, which means that the user has to estimate the length himself. In general a length of 1000 suffices for most practical problems, however, for complicated regions it might be necessary to extend this length.

   Array KPROB contains information of the definition of the problem and the type of boundary conditions. KPROB is also a variable-length array, with the same requirements concerning its length as for KMESH.

   Array INTMAT is a fixed-length array with a fixed length of 5. It contains information of the type of solution method that is used.

   In order to make it easy to change the length of the arrays KPROB and KMESH, these lengths have been stored in a parameter statement.

4. Data statement. This statement initializes the 4 arrays to zero. Strictly speaking this initialization is not necessary, but for security it is recommended.

5. Array ISOL is also a fixed-length array (length 5). It contains information of the solution.

   For arrays with user defined length, SEPRAN checks if these length suffice. The user must give in the first position of these arrays the value of the declared length.
   Furthermore position 3 of these arrays must be set to zero.

6. CALL SEPSTR (4.2.1).
   By the call of this subroutine, the SEPRAN program is started, the mesh is read as well as the problem definition and the type of solver to be used. SEPSTR (4.2.1) is one of the possible starting subroutines. See Chapter 4 for a complete overview of the various starting subroutines.

7. CALL PRESDF (5.2.1). The call of PRESDF defines the prescribed degrees of freedom. See Chapter 5 for more details.

8. CALL LINSOL (8.2.1).
   By the call of LINSOL (8.2.1) the coefficients of the problem to be solved are read, the system of equations is formed and solved.

   LINSOL (8.2.1) may be considered as a subroutine calling itself three other types of subroutines:

   Subroutines to fill the coefficients, see Chapter 6.

   Subroutines to build the system of equations, see Chapter 7.

   Subroutines to solve the system of equations, see Chapter 8.

   In complicated problems LINSOL (8.2.1) may be replaced by a series of subroutines, possibly in a complex loop. In fact LINSOL (8.2.1) represents the complete computational part.

9. CALL OUTPUT.
   The call of OUTPUT initiates the computation of derived quantities. Furthermore the solution, the derived quantities etcetera are written to a file. See Chapter 20.

10. CALL FINISH.
    This statement closes the SEPRAN computations. All SEPRAN files are closed and temporary files are deleted. It is recommended to use this statement always at the end of a SEPRAN program.

11. END statement, standard FORTRAN 77.

12. extra subroutines If extra (function) subroutines are required, they may be positioned at the end of the file containing the main program. However, it is also possible to put these subroutines in separate files.

2.2 Programming considerations

SEPRAN consists of a set of FORTRAN subroutines that can be used in standard FORTRAN 77 programs. The user has to write his own main program, in which the arrays must be declared. The following remarks are important:

- In the main program and each subroutine the first statement after the program or subroutine statement should be:

  ```
  IMPLICIT NONE
  ```

  which forces you to declare all variables explicitly. Furthermore, all reals must be declared double precision, because SEPRAN computes only in double precision in order to avoid loss of accuracy. Only at 64 bits computers that treat reals as 16 digit numbers, single precision should be used. This is for example the case on a CRAY computer but not on a 64-bit Silicon Graphics.

- Real constants must be used in double precision mode, i.e. you should use 3.5d0 instead of 3.5.

- In this manual all arrays or variables that are not explicitly declared satisfy the property that they are integer if there first letter is a letter from the range I-N and a double precision otherwise.

- All arrays are declared one-dimensional. In the main program the total space of arrays that are used in subroutines has to be allocated in an integer or double precision dimension statement. Since the FORTRAN compilers do not check the boundaries of the arrays, this enables us to declare arrays in subroutines by:

  ```
  DOUBLE PRECISION ARRAY(*)
        or
  INTEGER IARRAY(*)
  ```

  Therefore the array length does not have to be used in the parameter list. Of course in this way one has to be careful when declaring arrays. That is why for SEPRAN arrays a checking procedure is used.

- The first position of all SEPRAN arrays of variable length must contain the declared length (to be filled by the user). This length is checked during computation.
  The third position must be set equal to zero.

- For non-interactive systems the following FORTRAN conventions are standard:

  A C or an asterix (∗) in column 1 means a comment line.
  For other lines a symbol in column 6 means a continuation line, i.e. the statement of the preceding line is continued on this line. All statements should start after column 6, column 72 is the last column to be used.

- The input from the standard input file (for example a card reader, or a file) is organized in records. A record is a card (card reader) or a line (file). Records must always be at most 80 positions long.
  SEPRAN requires a special form of input as described in Section 1.4 of the Users Manual.

2.3 Linking and debugging programs

As soon as you write your own programs it is necessary to compile and link these subroutines. Linking implies that all SEPRAN subroutines as well as some general subroutines like mathematical subroutines from the library, i/o subroutines and so on, must be linked to the main program in order to get an executable.

SEPRAN provides some tools for the linking and compilation of your programs. These tools itself are machine-dependent, however, their use is the same for all computers.
If a user has written his own main program and possible extra subroutines, he may put them all in one file, or he may put subroutines in separate files for example one file per subroutine. Each of these files must have the extension .f in a unix environment or .for in a DOS environment. In fact it is also possible to link c subroutines, but this is too complicated to treat in this manual, especially since this is very machine-dependent.
Linking and compilation of the main program and subroutines may be performed in one call by the command seplink:

seplink *file subroutine1.f subroutine2.f ..*

where *file* is the name of the file containing the program without the extension .f and *subroutine1.f, subroutine2.f ..* are the names of extra subroutines to be compiled if they exist.
For example the command:

seplink example subroutine1.f

compiles the files example.f and subroutine1.f and links them into an executable example.

In the first step of seplink the fortran code is checked and translated. Fortran error-messages appear on the screen but in the case of DOS environment they are also written to a file named out.put so that they can be read afterwards. This is especially useful if the number of errors exceeds the size of one screen. The file out.put can always be inspected with a standard text editor. If the compilation has been carried out correctly, seplink links the program and subroutines with the SEPRAN library. seplink automatically links all subroutines in the directory that have been precompiled, i.e. all files with the extension .o in unix or .obj in msdos. So the user must remove these files if they should not be included in the executable. If one or more subroutines are missing seplink reacts with some machine-dependent message. For example in unix a common one is the message undefined symbol, followed by the name of the subroutine(s) provided with an underscore at the end of the name. For example if you did provide a function subroutine funbc instead of funcbc you get the message

undefined symbol

funcbc_

In a MSDOS (FTN77) environment the message is:

```
file example
The following routines are missing:-
FUNCBC
```

The error message "subroutines missing" usually results from an incorrectly spelled subroutine name or from the omission to declare an array.

Error messages of the linking phase are written directly to the screen and in case of a DOS environment also to a file named link.out so that they can be inspected afterwards. If both compilation and linking have been carried out succesfully seplink produces a file with the name of the first seplink parameter (that is without the extension .f). So seplink example subroutine1.f produces a file example. To run the program example in a unix environment you type:

example < inputfile > outputfile

or
example < inputfile

In outputfile the results of program example are written. These may be error messages of SEPRAN or output written by the user. If outputfile is omitted all information is written to the screen.

In a msdos environment you type:

example inputfile outputfile
or
example inputfile
or even
example

If output file is omitted the output is written to the file sepran.out, if the input file is also omitted the program reads the input from the file sepran.dat or produces an error message if that file does not exist.

It is also possible to compile subroutines before the linking phase. This may be done by the command compile:

compile *subroutine1.f subroutine2.f ..*

where *subroutine1.f, subroutine2.f ..* are the names of the subroutines to be compiled. So the following sequence may replace the first seplink command:

compile *subroutine1.f subroutine2.f ..*
seplink *file*

This possibility has the advantage that only subroutines that are changed must be compiled again since all object files are automatically linked with seplink.

If the user wants to utilize the debugger corresponding to the FORTRAN compiler at the computer system, he must use the debug equivalents of seplink and compile:

```
sepdbg
compiledbg
```

These commands have exactly the same meaning as seplink and compile, except that they use the debug option. If you do not know how to debug at your computer system, please inform at your local installation officer.

3 Mesh generation

3.1 Introduction

In this chapter all possibilities of generating a mesh by SEPRAN will be considered.
The first and most simple way of generating a mesh is to use program SEPMESH as described in Sections 2.1 and 2.2 of the Users Manual. If SEPMESH is used, a file meshoutput (unix) or a file meshout.put (msdos) is created that can be read by program SEPCOMP or alternatively by some of the SEPRAN start subroutines that are treated in Chapter 4.
Another possibility is to create a mesh by some other external mesh generator. In order to read such a mesh the user may utilize program SEPMESH in combination with the option USER and subroutine MESHUS as described in the Users Manual Section 2.4.6. This possibility requires of course some effort of the user.
A more suitable way of reading a mesh created by an external mesh generator is to transform this mesh into the formats that SEPRAN recognizes. In Section 3.3 the layout and formats of the file meshoutput are given.
All these options assume that the mesh is created first in order to run program SEPCOMP or a similar program. However, it also possible to create the mesh immediately in the main program. To that end subroutine MESH (3.2) is available. A simple reason to use subroutine MESH instead of program SEPMESH is that the user might want to change the mesh during the computations or perhaps the user wants one program instead of the series sepmesh, sepcomp, seppost. Subroutine MESH is not only capable of making a mesh but also of changing an existing mesh. This subroutine may read the standard mesh information for program SEPMESH but may also get all information through the parameter list. This last option, although complicated may be used if input of the standard input file should be avoided.
Sometimes the user wants to refine an existing mesh. This may be done globally, i.e. a complete uniform refinement is applied, or locally based on some criterion. SEPRAN provides a number of refinement subroutines. These subroutines may be used in combination with an adaptive algorithm. The new mesh created may be written to the file meshoutput with the aid of subroutine MESHWR (3.5). To read a mesh other then through the standard start subroutines, subroutine MESHRD (3.4) is available.
For a moving boundary problem or a free-surface problem it may be necessary to adapt the mesh in each step. Subroutine MESH may be used where all input is given through the parameter list. However, the creation of the input is quite complicated. To that end subroutine ADAPBOUN (3.9) has been developed, which adapts the boundaries. After that subroutine MESH may be called to create the new mesh.
A more automatic way of adapting the mesh is available by subroutine ADAPMESH (3.10, which in fact consists of a sequence of calls to ADAPBOUN (3.9.1 as well as MESH. Besides that, if necessary the problem definition and matrix structure are adapted.

In this chapter we deal with the following paragraphs:

3.2 treats subroutine MESH, the general mesh generator of SEPRAN. Especially the input through the parameter list is described completely.

3.3 describes the layout and formats of the file meshoutput.

3.4 deals with subroutine MESHRD (3.4), which reads the mesh stored in the file meshoutput.

3.5 treats subroutine MESHWR (3.5, which may be used to write a mesh to the file meshoutput.

3.6 treats subroutine REFINE which may be used to refine a mesh in a uniform way. This is equivalent to the option REFINE n TIMES in the input file as described in the Users Manual Section 2.2.

3.7 treats subroutine REFLOC for a local refinement.

3.8 deals with subroutine TRANSF. This subroutine may be applied to create a mesh consisting of other types of elements using an existing mesh. This is equivalent to the option TRANSFORM in the input file as described in the Users Manual Section 2.2.

3.9 describes subroutine ADAPBOUN (3.9.1 for the adaptation of a boundary.

3.10 describes subroutine ADAPMESH (3.10 for the adaptation of a mesh.

3.11 treats subroutine FEMESH (3.11. This subroutine is used to create a linear finite element mesh from a spectral mesh.

3.12 describes subroutine DEFMESH (3.12. This subroutine is used to change the co-ordinates of a mesh due to displacements. The difference with ADAPMESH is that the complete mesh is changed according to the solution vector and not only the boundary followed by an interpolation procedure as is the case in ADAPMESH. spectral mesh.

3.2 Creating a mesh by subroutine MESH

Description

Subroutine to generate a new mesh or to update an existing mesh. The information of the mesh is stored in array KMESH.

If subroutine MESH is used, SEPRAN may not be started with the starting subroutines SEPSTx, but these subroutines must be replaced by a series of calls. See Section 4.1 for the details.

Call

```
CALL MESH ( ICHOICE, IINPUT, RINPUT, KMESH )
```

Parameters

INTEGER ICHOICE, IINPUT(∗), KMESH(∗)

DOUBLE PRECISION RINPUT(∗)

ICHOICE Choice parameter indicating which actions must be performed by subroutine MESH. Possibilities:

- 0 All input is read from the standard input file. The arrays IINPUT and RINPUT are not used.
- 1 All input must be stored by the user in the arrays IINPUT and RINPUT.
- 2 All input is read from the standard input file and stored in the arrays IINPUT and RINPUT.
- 3 The mesh generator MESH has been called before. The mesh description (points, curves, etc.) remains the same, only the co-ordinates are changed. The numbering and the triangulation remain unchanged. The arrays IINPUT and RINPUT may be filled by a preceding call of MESH with ICHOICE=2.
- 4 All input is read from the arrays IINPUT and RINPUT. The co-ordinates and the topological description of the mesh are changed, however, the input as stored in IINPUT and RINPUT is the same as in a preceding call of subroutine MESH. The number of points and curves remains the same, as well as the number of elements along the curves. The co-ordinates of the user points and the curves must have been changed by a call of subroutine CHANBN (3.9.3 (Section 3.9) and, as a consequence, the number of elements in the surface may be changed. This possibility must be preceded by calls of subroutine MESH and subroutine CHANBN (3.9.3.
- 5 See ICHOICE=4, however, the topological description remains unchanged.
- 6 All input is read from the standard input file and stored in the arrays IINPUT and RINPUT. For this special value of ICHOICE no mesh is created, nor is array KMESH filled. The only purpose of the call of MESH with ICHOICE = 6, is the filling of the arrays IINPUT and RINPUT.
- 10 It is supposed that already a mesh has been created and that the boundary of the mesh has been adapted by subroutine ADAPBN (3.9.2 (Section 3.9).
  The co-ordinates and the topological description of the mesh are changed, The number of points and curves remains the same, as well as the number of elements along the curves. The number of elements in the surface may be changed.
- 11 See ICHOICE=10, however, the topological description remains unchanged.

IINPUT This array is not used if ICHOICE = 0.
If ICHOICE = 1, 3, 4 or 5 IINPUT must have been filled.
If ICHOICE = 2 or 6 IINPUT will be filled by subroutine MESH.
The structure of array IINPUT is as follows:

Pos. 1 : Declared length; see Users Manual

Pos. 2 : not used

Pos. 3 : in this position the actual length is set by SEPRAN. It is advised to initialize this position by 0.

Pos. 4-5 : are not used.

Pos. 6 : NDIM (dimension of space)

Pos. 7 : NUSPNT (number of user points)

Pos. 8 : NCURVS (number of curves)

Pos. 9 : NSURFS (number of surfaces)

Pos. 10 : NVOLMS (number of volumes)

Pos. 11 : NLINES (number of parts where line elements are created)

Pos. 12 : NSUREL (number of parts where surface elements are created)

Pos. 13 : NVOLEL (number of parts where volume elements are created)

Pos. 14 : IPLOT Indication whether a plot must be made. Possibilities:

0 : No plot is made.

i>0 : Plots are made of the curves and the surfaces. The value of IPLOT must be equal to JMARK + 10 × NUMSUB + 1 with

NUMSUB : the submeshes with numbers ≤ NUMSUB are not plotted.

JMARK : Indication of how the plot of the mesh must be made. Possibilities:

0,3 : Each nodal point is marked with a star and its nodal point number.

1,4 : Each nodal point is marked with a star. Nodal point numbers are **not** plotted.

2,5 : Nodal points are not marked, nor are nodal point numbers plotted.

When JMARK < 3 all element numbers are plotted in the centroid of the elements, when JMARK ≥ 3 no element numbers are plotted.

i< 0 : If IPLOT < 0, minus IPLOT gives the starting address of information about plotting.
Let ISTART = -IPLOT be this starting address. Then the following information is stored from IINPUT(ISTART):

IINPUT(ISTART) : JMARK
IINPUT(ISTART+1) : NUMSUB
IINPUT(ISTART+2) : PLOTCURV
IINPUT(ISTART+3) : PLOTCURVNOD
IINPUT(ISTART+4) : PLOTUSERP
IINPUT(ISTART+5) : COLOUR
IINPUT(ISTART+6) : PLOT3DMESH
IINPUT(ISTART+7) : RENUM_PLOT
IINPUT(ISTART+8) : NOMESH
IINPUT(ISTART+9) : ORIENTATION

with JMARK and NUMSUB as defined before.

PLOTCURV indicates if the curves must be plotted in a separate picture. Possible values:

0 Curves are not plotted.

1 Curves are plotted without curve numbers.

2 Curves are plotted provided with curve numbers.

Default value: 2.

PLOTCURVNOD indicates if nodes along the curves must be plotted in the picture containing the curves. So this parameter only makes sense for $c > 0$. Possible values:

0 Nodes are not plotted.

1 Each node is indicated with a cross-symbol.

2 Each node is indicated with a symbol from the symbol table. The sequence number in the symbol table is equal to PLOTCURVNOD – 1.
Which symbols are stored in the symbol table depend on your plotting package.

Default value: 0.

PLOTUSERP indicates if user points must be plotted in a separate plot. Possible values:

0 User points are not plotted.

1 Each user point is indicated with a cross but not with a number

2 Each user point is indicated with a cross and provided with its user point sequence number.

Default value: 0.

COLOUR indicates if colors must be used to distinguish certain quantities. The most important object of the parameter is to plot each element group with a different color in the final plot of the mesh. Possible values:

0 Only one color (the standard color) is used.

1 The default colors are used. Each element group gets a different color.

2 Colors are used. Each element group gets a different color. The first element group gets color cl, the next one cl+1, etc.

Default value: 0.

PLOT3DMESH indicates if the final 3D mesh with hidden lines must be plotted (PLOT3DMESH = 1) or not (PLOT3DMESH = 0).

RENUM_PLOT indicates if an extra plot with renumbered nodal points must be made (1) or not. See User manual Section 2.2.

NOMESH indicates if a plot of the mesh must be made(0) or not (1)

ORIENTATION is only used in case of a 3D plot with hidden lines. It indicates the orientation of the axis. Possible values:

1 Standard x-y-z orientation

2 z-x-y orientation

3 y-z-x orientation

Pos. 15 : IRENUMBER. When IRENUMBER = 0, no renumbering is performed, when IRENUMBER > 0, the nodal points are renumbered internally in order to get an optimal "profile", or "band width". Possible values for IRENUMBER are

1 The Cuthill-Mckee (1969) [1] algorithm is used in order to optimize the profile. If the profile of the original matrix is smaller than that of the "Cuthill-Mckee" renumbered matrix, then no renumbering takes place.

2 An algorithm due to Sloan (1986) [2] is used in order to optimize the profile. If the profile of the original matrix is smaller than that of the "Sloan" renumbered matrix, then no renumbering takes place.

3 Both the Sloan algorithm and Cuthill-Mckee are used to compute a renumbering. The final renumbering is based on the smallest profile of one of these algorithms and the original numbering.

4 The Cuthill-Mckee algorithm is used in order to optimize the band-width. If the band-width of the original matrix is smaller than that of the "Cuthill-Mckee" renumbered matrix, then no renumbering takes place.

5 An algorithm due to Sloan is used in order to optimize the band-width. If the band-width of the original matrix is smaller than that of the "Sloan" renumbered matrix, then no renumbering takes place.

6 Both the Sloan algorithm and Cuthill-Mckee are used to compute a renumbering. The final renumbering is based on the smallest band-width of one of these algorithms and the original numbering.

7 The Cuthill-Mckee algorithm is used to compute a renumbering independent of the relation with the original numbering.

8 The Sloan algorithm is used to compute a renumbering independent of the relation with the original numbering.

Pos. 16 JCOARS Indication whether coarseness is used (JCOARS=1) or not (JCOARS=0).

Pos. 17 JCONNECT Indication that elements are defined that connect user points or nodal points on curves or surfaces (1) or not (0).

Pos. 18 JMAXIN With this parameter it is indicated if the maximum number of nodal points, curves, surfaces or volumes is given by their default values, or that these values are given explicitly.
If JMAXIN = 0, the default values are used. These default values are:

maximum number of user points: 1000

maximum number of curves: 1000

maximum number of surfaces: 1000

maximum number of volumes: 500

If JMAXIN > 0, the value of JMAXIN indicates the position where the maxima are put in array IINPUT. These positions must be put after the information given below. So the maximum values must be stored as follows:

IINPUT(JMAXIN): maximum number of user points

IINPUT(JMAXIN+1): maximum number of curves

IINPUT(JMAXIN+2): maximum number of surfaces

IINPUT(JMAXIN+3): maximum number of volumes

Pos. 19 JEXTRA With this parameter it is indicated if extra points must be created for some or all of the elements. This position corresponds to the COMMAND INTERMEDIATE POINTS defined in the Users Manual Section 2.2. If JEXTRA=0, no intermediate points are defined. If JEXTRA > 0, the value of JEXTRA defines the starting position of information for the extra intermediate points. See the heading "INTERMEDIATE POINTS" for extra information.

Pos. 20 JLAST With this parameter 20 extra positions are reserved with information about some extra possibilities. This information is stored from position JLAST.

Next from position 21: information of curves; curves must be stored sequentially.
For each curve: (NCURVS curves)

(i) ICURVE code for type of curve to be generated; possibilities:

1 straight line.

2 user defined curve.

3 part of a circle.

4 spline function.

5 straight line with coarseness.

6 arc with coarseness.

7 curve consisting of curves.

8 curve defined by subroutine FUNCCV (parameter function).

9 curve defined by subroutine FUNCCV (parameter function), with coarseness.

10 curve constructed by translation of other curve with smaller curve number.

11 curve constructed by translation and rotation of other curve with smaller curve number (both curves must be situated in a plane).

12 spline defined with coarseness.
13 curve constructed by the command reflect of other curve with smaller curve number.
14 spline defined by a number of curves. The subcurves are made obsolete.
15 profile.
16 cprofile.
17 circle.
18 4 parts of a circle
19 4 half arcs from circle to upper or lower point of sphere
20 user defined curve by subroutine OWN_curve
21 frame curve, a special option for PLAXIS only
22 ell_arc
23 cell_arc

(ii) ISHAPE code for type of elements to be created along the curve; possibilities:

1 linear elements.
2 quadratic elements.

when ICURVE = 7,10, 11 or 12 this position is filled by subroutine MESH. (value 1).

(iii) NELM number of elements to be created. When ICURVE = 5, 6,7,9,10,11,12 or 16 this position is filled by subroutine MESH.

(iv) ... The next positions depend on the value of ICURVE.

ICURVE=1,2,3,17,22

(iv) IRATIO code for the distribution of the nodal points; possibilities:

0 equidistant mesh size.
1 the last element is FACTOR times the first one.
2 each next element is FACTOR times the preceding one.

(v),(vi), . . . points defining the curve.
The number of points for ICURVE = 1 is 2, for ICURVE = 2 is NELM+1, for ICURVE = 3 is 3 and for ICURVE = 17 is NDIM.

ICURVE=4,12

(iv) IRATIO (See above, this value is only used for ICURVE=4)

(v) ITYPE type of spline to be created.
The parameter ITYPE consists of two parts ITYPE_type and IALPHA according to ITYPE = ITYPE_type + 10 × IALPHA, with ITYPE_type the value of TYPE as defined in the Users Manual Section 2.3 and IALPHA indicating how the factor α is defined. Possible values for IALPHA:

0 $\alpha = 1$
1 $\alpha = 0$
2 $\alpha = 0.5$
3 α is stored in array RINPUT in the same position where the parameter t_0 must be placed if ICURVE = 8 or 9.

(vi) NDEFCV number of points defining the curve.

(vii),(viii), . . . points defining the curve.

followed by:
first tangent node
second tangent node
for ICURVE = 12, this must be followed by:
indication whether the number of end points of elements on the curve is even (2) odd (3) or free (0,1).

ICURVE=5

(iv) -

(v),(vi) points defining the curve.

(vii) indication whether the number of end points of elements on the curve is even (2) odd (3) or free (0,1).

ICURVE=6, 23

(iv) -

(v),(vi),(vii) points defining the curve.

(vii) indication whether the number of end points of elements on the curve is even (2) odd (3) or free (0,1).

ICURVE=7

(iv) the number of curves from which the curve is defined.

(v),(vi),(vii),... curves defining the curve.

ICURVE=8

(iv) IRATIO, see above

(v),(vi) initial respectively end point of curve.

ICURVE=9

(iv) -

(v),(vi) initial respectively end point of curve.

(vii) indication whether the number of end points of elements on the curve is even (2) odd (3) or free (0,1).

ICURVE=10,11,13

(iv) number of user points that are given on this curve.

(v) curve number of curve from which the actual curve is copied.

(vi),(vii),... user point numbers of points on the curve.

ICURVE=14

(iv) IRATIO with respect to the definition of the spline. At this moment IRATIO=2 is assumed.

(v) User point number of first point at curve.

(vi) User point number of last point at curve.

(vii) the number of curves from which the curve is defined.

(viii),(ix),(x),... curves defining the curve.

ICURVE=15

(iv) IRATIO See ICURVE = 1

(v) IPROFILE, defines the shape of the profile At this moment the following values for IPROFILE are available:

1. upper_naca0012
2. lower_naca0012

(vi),(vii) points defining the curve.

ICURVE=16

(iv) -

(v) IPROFILE, defines the shape of the profile, see ICURVE=15

(vi),(vii)

(viii) indication whether the number of end points of elements on the curve is even (2) odd (3) or free (0,1).

ICURVE=20

(iv) IFUNC, is used to distinguish between several possibilities.

(v),(vi) 2 end points (user points) of the curve.

Next positions when NSURFS > 0:
information of surfaces; surfaces must be read sequentially.
For each surface: (NSURFS surfaces)

(i) ISURF code for type of mesh generator to be used; possibilities:

1 RECTANGLE (see Users Manual 2.4.2).
2 GENERAL (see Users Manual 2.4.1).
3 TRIANGLE (see Users Manual 2.4.7).
4 QUADRILATERAL (see Users Manual 2.4.3).
5 MESHUS: User defined mesh generator (see Users Manual 2.4.6).
7 SURFACES: the surface consists of surfaces.
8 TRANSLATE: the surface is constructed by translation of preceding surface with smaller surface number.
9 ROTATE: the surface is constructed by translation and rotation of preceding surface with smaller surface number.
10 PIPESURFACE: cylinder type surface.(see Users Manual 2.4.5).
11 SURFACES: the surface consists of ordered subsurfaces.
Remark: if all subsurfaces are of the type pipe surface, then internally ISURF gets the value 12.
13 COONS (see Users Manual 2.4.4).
14 SIMILAR: the surface is constructed by a similarity transformation of preceding surface with smaller surface number.
15 PARSURF: the surface is defined as parameter surface (see Users Manual 2.4.8)
16 REFLECT: the surface is constructed as a reflection of a surface with smaller surface number.
17 RECTANGLE with the option smoothing applied (see Users Manual 2.4.2).
18 ISOPAR: the surface is constructed by the submesh generator ISOPAR (see Users Manual 2.4.9).
19 PAVER: the surface is constructed by the submesh generator PAVER (see Users Manual 2.4.10).
20 SPHERE: the surface is constructed by several calls to the submesh generator ISOPAR in order to create a sphere.
At this moment it can only be used if the mesh input is read by sepmesh, which means that the user did not fill iinput/rinput himself.
21 COPY_SURFACE: the surface is a copy of another surface
22 FRAMESURF: special purpose surface generator as described in the Users Manual Section 2.4.12
23 TRIANGLE with internal points and/or internal curves. This is the extension of TRIANGLE (3), with fixed internal user points or curves.

(ii) ISHAPE Code for type of elements to be created on the surface; Possibilities see Users Manual Table 2.1.1.
When ISURF = 7,8, 9 or 21 this position is filled by subroutine MESH.
The next positions depend on the value of ISURF.

ISURF=1

(iii) NDEFSF number of curves defining the surface.
(iv),(v),...,(iii)+NDEFSF NDEFSF curve numbers.
(iii)+NDEFSF+1 N
(iii)+NDEFSF+2 M

ISURF=2,3,15,19

(iii) NDEFSF number of curves defining the surface.

(iv),(v),...,(iii)+NDEFSF NDEFSF curve numbers.

ISURF=4, 13

(iii) NDEFSF number of curves defining the surface.

(iv),(v),...,(iii)+NDEFSF NDEFSF curve numbers.

(iii)+NDEFSF+1 IBLEND

(iii)+NDEFSF+2 ICURVE These parameters indicate how QUADRILATERAL or COONS create the internal nodes and elements. See Users Manual 2.4.

(iii)+NDEFSF+3 Type of mapping from 2d to 3D (0=standard, 1 = sphere)

(iii)+NDEFSF+4 User point number corresponding to center of sphere in case mapping=1

The last two positions only for ISURF = 13

ISURF=5

(iii) NDEFSF number of curves defining the surface.

(iv),(v),...,(iii)+NDEFSF NDEFSF curve numbers.

(iii)+NDEFSF+1 NELEM

(iii)+NDEFSF+2 NPOINT

ISURF=6

(iii) NDEFSF number of curves defining the surface.

(iv),(v),...,(iii)+NDEFSF NDEFSF curve numbers.

(iii)+NDEFSF+1 FILE

(iii)+NDEFSF+2 NELEM

(iii)+NDEFSF+3 NPOINT

ISURF=7

(iii) NDEFSF number of surfaces defining the surface.

(iv),(v),...,(iii)+NDEFSF NDEFSF surface numbers.

ISURF=8, 9, 14, 16

(iii) NDEFSF number of curves defining the surface.

(iv),(v),...,(iii)+NDEFSF NDEFSF curve numbers.

(iii)+NDEFSF+1 surface number of surface to be copied

ISURF=10

(iii) NDEFSF consists of two parts NDEFSF_ORIG and IDIAG, according to: NDEFSF = NDEFSF_ORIG + 10000 × (IDIAG_ORIG + 10 × INTERPOL). NDEFSF_ORIG is the standard definition of NDEFSF and IDIAG_ORIG refers to the parameter DIAGONAL_DIR in the input, as described in the Users Manual Section 2.4.5.

The parameter INTERPOL defines the type of interpolation used, see Users Manual Section 2.4.5.

Possible values:

0 Standard (default) interpolation.

1 Horizontal straight lines.

2 Vertical straight lines.

(iv),(v),...,(iii)+NDEFSF NDEFSF curve numbers.

ISURF=11, 12

(iii) NROW number of rows of subsurfaces defining the surface, followed by:

(iv),(v),...,(iii)+NROW NDEFSF surface numbers. For each row the number of subsurfaces in that row.

(iii)+NROW+1,(iii)+NROW+2,...,(iii)+NROW+NDEFSF NDEFSF surface numbers in the sequence given by each row.

ISURF=17

(iii) NDEFSF number of curves defining the surface.

(iv),(v),...,(iii)+NDEFSF NDEFSF curve numbers.

(iii)+NDEFSF+1 N

(iii)+NDEFSF+2 M

(iii)+NDEFSF+3 ISMOOTH (smoothing factor)

(iii)+NDEFSF+4 0 (Not yet used)

ISURF=18

(iii) NDEFSF number of curves defining the surface.

(iv),(v),...,(iii)+NDEFSF NDEFSF curve numbers.

(iii)+NDEFSF+1 Type of mapping from 2d to 3D (0=standard, 1 = sphere)

(iii)+NDEFSF+2 User point number corresponding to center of sphere in case mapping=1

(iii)+NDEFSF+3 Indicates how the "triangle" must be subdivided (0=standard, 1 = regular)

ISURF=20

(iii) NDEFSF number of curves defining the surface (1)

(iv) Generating curve

(v) User point number of centroid of sphere

(vi) ITYPE, i.e. type of sphere.
Possible values:

1. Complete sphere
2. Upper_half sphere
3. Lower_half sphere

(vii) Surface number of actual surface

(viii) Curve number of actual curve corresponding to circle

(ix.(x) sequence numbers of subsurfaces

ISURF=21

(iii) Surface sequence number of surface to be copied

(iv) Surface sequence number of surface to be copied

ISURF=22, 23

(iii) NDEFSF number of curves defining the surface.

(iv),(v),...,(iii)+NDEFSF NDEFSF curve numbers.

NDEFSF+4 NINTERNAL_POINTS, i.e. number of internal points that are fixed by prescribing user points numbers. See Users Manual Section 2.4.7.

NDEFSF+5 NINTERNAL_CURVES, i.e. number of internal curves that are given. See Users Manual Section 2.4.7.

NDEFSF+6 ... NDEFSF+5+NINTERNAL_POINTS user point numbers of internal points.

NDEFSF+6+NINTERNAL_POINTS ... NDEFSF+5+NINTERNAL_CURVES curve numbers of internal curves.

Next positions when NVOLMS > 0:
information of volumes; volumes must be read sequentially.
For each volume: (NVOLMS volumes)

(i) IVOLM code for type of mesh generator to be used; possibilities:

1 BRICK (see Users Manual 2.5.1).

2 USER user defined mesh generator (see Users Manual 2.4.6).

3 INPUT

4 PIPE: PIPE (see Users Manual 2.5.2).

5 GENERAL (see Users Manual 2.5.4).
6 TRANSLATE (see Users Manual 2.5).
7 ROTATE (see Users Manual 2.5).
8 REFLECT (see Users Manual 2.5).
9 CHANNEL (see Users Manual 2.5.3).
10 GENERAL, with internal points, curves and/or surfaces (see Users Manual 2.5.4). This is the extension of GENERAL(4) with the fixed internal quantities.

(ii) ISHAPE Code for type of elements to be created on the volume; Possibilities see Users Manual Table 2.1.1.

(iii) NDEFVL number of surfaces defining the volume, followed by:

(iv),(v),...,(iii)+NDEFVL NDEFVL surface numbers. These numbers are followed by the extra information necessary for the submesh generators: When

IVOLM = 1 : N,M,L
IVOLM = 2 : NELEM,NPOINT
IVOLM = 3 : FILE,NELEM,NPOINT
IVOLM = 4 : -
IVOLM = 5 : -
IVOLM = 6 : volume number of volume to be copied
IVOLM = 7 : volume number of volume to be copied
IVOLM = 8 : volume number of volume to be copied
IVOLM = 9 : -
IVOLM = 10 : NINTERNAL_POINTS, NINTERNAL_CURVES, NINTERNAL_SURFACES, followed by the corresponding user points, curve numbers and surface numbers in that sequence.
These extra positions refer to the internal quantities as defined in the user manual Section 2.5.4.

Next positions when NLINES > 0:
information of the point or line elements sequentially.
For each group of line elements:

(i) ISHAPE shape number defining the number of points in the element minus one.
When ISHAPE = -1 one element is created consisting of all nodal points on the curves IFIRST, . . . ,ILAST
When ISHAPE = 0, the user points IFIRST to ILAST are used as point element.

(ii) IELGRP element group number.

(iii) IFIRST first curve number for line elements of this group.

(iv) ILAST last curve number for line elements of this group.

Next positions when NSUREL > 0:
information of the surface elements sequentially.
For each group of surface elements:

(i) ISHAPE shape number for mesh generation.

(ii) IELGRP element group number.

(iii) IFIRST first surface number for surface numbers of this group.

(iv) ILAST last surface number for surface numbers of this group.

Next positions when NVOLEL > 0:
information of the volume elements sequentially.
For each group of volume elements:

(i) ISHAPE shape number for mesh generation.

(ii) IELGRP element group number.

(iii) IFIRST first volume number for volume numbers of this group.

(iv) ILAST last volume number for volume numbers of this group.

Next positions when JCONNECT = 1:

(i) NCONUSERPOINTS Number of pairs of user points that are connected.

(ii) NCONCURVES Number of pairs of curves that are connected.

(iii) NCONSURFS Number of surfaces that are connected.

Next positions: when NCONUSERPOINTS > 0.
For each pair of user points:

(i) IELGRP element group number.

(ii) IFIRST first user point.

(iii) ILAST last user point.

Next positions: when NCONCURVES > 0.
For each pair of curves:

(i) IELGRP element group number.

(ii) ITYPE indication which nodal points are connected. See the parameter l in the Users Manual 2.1

(iii) IFIRST first curve number.

(iv) ILAST last curve number.

Next positions: when NCONSURFS > 0.
For each pair of surfaces:

(i) IELGRP element group number.

(ii) ITYPE indication which nodal points are connected. See the parameter j in the Users Manual 2.1

(iii) IFIRST first surface number.

(iv) ILAST last surface number.

(v) IEXCLUDE Indicates which curves are excluded. Possible values

- 0 : no curves are excluded
- -1 : all outer curves of the first surface are excluded
- n>0 : there are n curves of the first surface that must be excluded
  In this case n extra positions are used to store the n curve numbers.

INTERMEDIATE POINTS

For the definition of the extra points it is necessary to set the starting position of the information in IINPUT(19). This extra information must be positioned after all information concerning curves, surfaces, volumes, and various types of elements.
The first position (IINPUT(JEXTRA)) contains the number (n) of parts for which information is stored.
At this moment only $n = 1$ is allowed. The next four positions contain :

Pos. 1 Number of extra nodes on a side of an element (non-negative)

Pos. 2 Number of internal points in a surface element. If this number is -1 the internal points are extended regularly from the boundary as described in the Users Manual Section 2.2.

Pos. 3 Number of internal points in a volume element. If this number is -1 the internal points are extended regularly from the boundary.

Pos. 4 Type of subdivision of extra points. Possibilities :

0 Nodes are positioned equidistantly

1 Nodes are positioned according to a Gauss-Legendre-Lobatto subdivision

Extra information if JLAST>0. The positions JLAST to JLAST+19 are reserved for the following quantities:

JLAST: IREFIN indicates how many times the mesh must be refined ($0 \leq$ IREFIN < 100).

JLAST+1: ITRANS indicates to what type of elements the mesh consisting of linear triangles and tetrahedrons must be transformed.

JLAST+2: ICHECK defines the check level as described in the Users Manual Section 2.2.

JLAST+3: ISKIPTOPOL, defines if topology must be computed (0) or not (1).

JLAST+4: ISTART_INTERFACE, defines starting address of information of interface elements. If 0 there are no interface elements.

JLAST+5: ISTART_PROPERTIES, defines starting address of integer and real properties. If 0 there are no properties connected to element groups.

JLAST+6: ISTART_CHANGE_COORDINATES, defines starting address of information about how to change the coordinates.
If 0 the coordinates must not be changed.

JLAST+7: ISTART_OBSTACLES, defines starting address of information about obstacles. If 0 no obstacles are present.

JLAST+8: IWRIT_FACES. If IWRIT_FACES = 1, the edges and in case of volumes also the faces are computed and written to the file meshoutput.
If IWRIT_FACES = 0 no edges or faces are written.

JLAST+9: ISCALE, defines if scaling is applied.
Possible values:

0 no scaling

1 scale always

2 scaling defined per surface or volume

JLAST+10: ITYPE_SCALE, defines type of scaling.
Possible values:

0 all Cartesian directions

1 x-direction only

2 y-direction only

3 z-direction only

4 auto-detect

JLAST+11: IPARALLEL, defines how the mesh must be subdivided in case of parallel computing.
At this moment IPARALLEL is an integer but in the future this may be a starting address, referring to an integer.
IPARALLEL consists of two parts: METHOD_PARALLEL and NUM_PROCESSORS according to METHOD_PARALLEL + 10 × NUM_PROCESSORS.

METHOD_PARALLEL defines the type of subdivision that is used Possible values:

0 no parallelization

1 each surface (2D) or volume (3D) is a parallel block

2 the region is subdivided into layers of elements each with approximately NPOINT/NUM_PROCESSORS nodes, where NPOINT is the number of nodes in the mesh.
NUM_PROCESSORS defines the number of processors that must be used.
This parameter is only used for METHOD_PARALLEL > 1.

JLAST+12: ISTART_DUMMY, defines the starting address of information about dummy element groups.

The other positions are not yet defined. The user must set them equal to zero.

If ISTART_INTERFACE > 0, extra positions are required to describe the interface elements. These positions start at position ISTART_INTERFACE.
Contents of this part:

Pos. 1 NPOININTERFACE Number of parts where point interface elements are defined. (must be zero at present).

Pos. 2 NCURVINTERFACE Number of parts where curve interface elements are defined.

Pos. 3 NSURFINTERFACE Number of parts where surface interface elements are defined.

Next NPOININTERFACE positions information about point interface elements.
Not yet available.

Next positions information about curve interface elements. The information is stored in NCURVINTERFACE groups of 4 positions each. These 4 positions contain:

1 IELGRP, element group sequence number

2 ISHAPE, shape number of elements, if 0 the default is used.

3 IFIRST, first curve number provided with sign.

4 ILAST, last curve number provided with sign.

Next positions information about surface interface elements. The information is stored in NSURFINTERFACE groups of 4 positions each. These 4 positions contain:

1 IELGRP, element group sequence number

2 ISHAPE, shape number of elements.
This shape number must be 70. Internally it is replace by 70+*oshape*, where *oshape* is the shape number of the elements in the two surfaces.

3 IFIRST, first surface number.

4 ILAST, second surface number.

If ISTART_PROPERTIES > 0, extra positions are required to describe the integer properties. These positions start at position ISTART_PROPERTIES.
Contents of this part:

Pos. 1 NINTPROP Number of integer properties per element group.

Pos. 2 NREALPROP Number of real properties per element group.

Next NELGRP × NINTPROP positions containing the integer properties, with NELGRP the number of element groups. These positions may be considered locally as a two dimensional array of size (1:NINTPROP,1:NELGRP).

If ISTART_CHANGE_COORDINATES > 0, extra positions are required to describe the information about changing the coordinates. These positions start at position ISTART_CHANGE_COORDINATES.
Contents of this part:

Pos. 1 NCHANCOOR, Number of parts where coordinates must be changed.

Must be followed by 4 × NCHANCOOR positions containing information about the changing of co-ordinates. Locally this part may be considered as an array of size (1:4,1:NCHANCOOR) with the following contents

Pos. (1,i) ifunc_chan. This is the parameter FUNC_CHAN as defined by the user in the functional description at input. See the Users Manual Section 2.2.

Pos. (2,i) itype, defines along what part of the mesh the coordinates must be changed and when they are changed.
This position consists of two parameters: ICHANGECR and IWHEN according to Pos(2,i) = ICHANGECR + 100 × IWHEN.
Possible values for ICHANGECR:

0 The complete mesh is used

1 The points IFIRST to ILAST are used

2 The curves IFIRST to ILAST are used

3 The surfaces IFIRST to ILAST are used

4 The volumes IFIRST to ILAST are used

Possible values for IWHEN:

0 The coordinates are changed at the end (AT_END)

1 The coordinates are changed immediately after creating (IMMEDIATELY)

Pos. (3,i) IFIRST, see pos (2,i).

Pos. (4,i) ILAST, see pos (2,i).

If ISTART_OBSTACLES > 0, extra positions are required to describe the information about the obstacles. These positions start at position ISTART_OBSTACLES.
Contents of this part:

Pos. 1 NUMOBST i.e. number of obstacles.

Must be followed by 2 × NUMOBST positions containing information about the obstacles. Locally this may be considered as a two dimensional array of size (1:2,1:NUMOBST) with the following contents:

Pos. (1,i) Type of obstacle.
Possible values:

1 The obstacle is defined by a closed curve.

Pos. (2,i) Curve sequence number of obstacle.

If ISTART_DUMMY > 0, extra positions are required to give information about the dummy element groups.
Contents of this part:

Pos. 1 NUMELGRPDUMMY i.e. number of dummy element groups.

Must be followed by 2 × NUMELGRPDUMMY positions containing for each dummy element group, the element group sequence number as well as the number of nodes for all elements in this group.

RINPUT This array is not used if ICHOICE = 0.
If ICHOICE = 1, 3, 4 or 5 RINPUT must have been filled.
If ICHOICE = 2 or 6 RINPUT will be filled by subroutine MESH.
The structure of array RINPUT is as follows:

Pos. 1 Declared length; see User Manual.

2-5 are not used, except pos. 3 where the actual length is set by SEPRAN.

6 PLOTFM or - SCALE

7 YFACT

8, . . . ,7+NDIM × **NUSPNT** co-ordinates of points in the sequence:
$x_1, y_1, z_1, x_2, y_2, z_2, ...$

When JCOARS = 1 then

8+NDIM × **NUSPNT** unit length.

9+NDIM × **NUSPNT . . .8+(NDIM+1)** × **NUSPNT** coarseness in the user points in the sequence: $c_1, c_2, ..., c_{NUSPNT}$

9+(NDIM+1) × **NUSPNT** MAXRATIO, see Users Manual.

Next NCURVS positions: for each curve the factor FACTOR.

Next 2 × NCURVS positions: for each curve the factors t_0 and t_1 in the sequence $t_0(1), t_1(1), t_0(2), t_1(2)$. These positions are only necessary when ICURVE=8 or 9 is used.

Next 4 × NSURFS positions: for each surface the factors $u1$, $u2$, $v1$ and $v2$ in that sequence. These positions are only used when ISURF= 15.

If the submesh generator PAVER is used, 4 × NCURVS extra positions are used to store the factors t and f for the curves Ci and -Ci.
Hence this extra array may locally be considered as a two-dimensional array of size 4 × NCURVS, with the following contents:

array(1,i) contains the factor t for curve -Ci.

array(2,i) contains the factor f for curve -Ci.

array(3,i) contains the factor t for curve Ci.

array(4,i) contains the factor f for curve Ci.

Next if NDIM = 3 and IPLOT ≠ 0 and PLOT3DMESH = 1 three extra positions are used containing the co-ordinates of the eye point (see the Users Manual Section 2.2).

Next if NREALPROP > 0 then NELGRP × NREALPROP positions containing the real properties, with NELGRP the number of element groups. These positions may be considered locally as a two dimensional array of size (1:NREALPROP,1:NELGRP).

KMESH In this array information of the mesh will be stored. This array has a variable length but in general a length of 100 positions is sufficient. The user must fill position 1 with the declared length. It is advised to set position 3 equal to 0.
For an exact definition of the contents of array KMESH the reader is referred to Section 24.2.

Input

ICHOICE must have a value.
Depending on the value of ICHOICE, input is read from the standard input file (ICHOICE = 0, 2, 6) or from the arrays IINPUT and RINPUT (ICHOICE = 1, 3, 4, 5).

Output

The mesh has been generated.
Array KMESH has been filled.
If required a plot of the mesh and submeshes has been made.
Depending on the parameter ICHOICE the arrays IINPUT and RINPUT may have been filled.

Warning If subroutine MESH is called a second time, where the arrays IINPUT and RINPUT have been filled by a first call of MESH and moreover, they have been kept unchanged, still the co-ordinates of the first and second mesh may differ in the fourth or fifth decimals. This is due to an iterative method that is used to improve the co-ordinates computed by the submesh generators GENERAL or TRIANGLE.

3.3 The file meshoutput

In this section we shall give an exact description of the file meshoutput. The exact name of the file is given in the SEPRAN environment file as described in the Users Manual Section 1.5.
In a unix environment this file is usually called meshoutput, whereas in MSDOS the file name meshout.put is most commonly used.
The meshoutput file corresponds to the parameter iref10 which is defined in the sepran.env file as *Unit number for mesh output file*. This file may be either formatted or unformatted depending on the line file 10: *xxx* in the sepran.env file, where *xxx* may be either *formatted* or *unformatted*.
Where we refer to a record in the file, actually one or more records may be used. Essential is that each time record is used it means in fact new record. In case of unformatted files usually records is identical to one record, in case of formatted files many times several records are used in order to increase readability.
The structure of the file is the same for formatted and unformatted files.

The global structure of the meshoutput file is as follows:

Records 1,2 General information about the mesh.

Records 3-5 3 tag records with text information

Next nelgrp+1 records General information about the topology if the number of element groups (nelgrp) is larger than 1.

Next records Co-ordinates of the mesh.

Next records Topology of the mesh.

Next records Nodal point sequence numbers of the user points.

Next records Information of the curves.

Next records Information of the surfaces

Next records Possible renumbering of nodal points

Next records KMESH part t, i.e. information about the elements with the highest dimension.

Next we will consider the structure in more detail:

Records 1,2 General information about the mesh.
This part consists of two records with the following contents:

Record 1 consists of 10 parameters:

Pos. 1: NDIM The absolute value of NDIM indicates the dimension of the space (1,2 or 3). If NDIM>0 a very old file meshoutput is used. In that case record 2 is not utilized.
For present files always record 2 is used and hence NDIM has been provided with a negative sign.

Pos. 2: NPELM Maximum number of points in all elements.

Pos. 3: NBOUN When NELGRP > 1, NBOUN $= 0$ should be used, otherwise NBOUN gives the number of nodal points on the boundary of an element. (NBOUN $\leq$ NPELM).
NBOUN is only used for the plotting of the mesh; it indicates how many nodal points must be connected when an element is plotted. In that case the first NBOUN points are connected and point NBOUN is connected with point 1.

Pos. 4: ISHAPE When NELGRP > 1, ISHAPE should be zero.
Otherwise it gives the shape number for mesh generation. Shape numbers are defined in the Users Manual Table 2.2.1, except in the case of connection elements when shape numbers in the range -10020 to -10001 are used. Consult the Users Manual Section 2.2.
When a new element is used, that does not belong to Table 2.2.1, ISHAPE = 0 should be used.
ISHAPE, NBOUN and NPELM may not conflict.

Pos. 5: NELGRP Number of element groups.

Pos. 6: NPOINT Number of nodal points.

Pos. 7: NELEM Number of elements.

Pos. 8: NUSPNT Number of user points.
User points do only have to be defined if they have some function in the computational program or the post-processing to indicate certain points.

Pos. 9: NCURVS Number of curves.
Curves do only have to be defined if they have some function in the computational program or the post-processing to indicate certain curves.

Pos. 10: NSURFS Number of surfaces.
Surfaces do only have to be defined if they have some function in the computational program or the post-processing to indicate certain surfaces.

Record 2 consists of 10 parameters

Pos. 1: NVOLMS Number of volumes.

Pos. 2: KELMT KMESH(36). This indicates if KMESH part t has been filled or not. For all new files meshoutput, array KELMT>0. If KELMT=0, KMESH part t is created by the read subroutine.

Pos. 3: NEXTPNT Number of extra points generated in case of nodes provided by the user.

Pos. 4: -

Pos. 5: KELMU If > 0, the information of the spectral elements is stored in file meshoutput.

Pos. 6: KELMJ If > 0, the renumbering of the nodal points is stored in file meshoutput.

Pos. 7: KELMM If > 0, information of the curves is stored in meshoutput.

Pos. 8: KELMN If > 0, information of the surfaces is stored in meshoutput.

Pos. 9: IMAP If 1 KMESH part h is written if 0 not.

Pos. 10: IVERS Version sequence number. The present version number is 8.

Next 3 records 3 records with text information.
Each record starts with `//`. The first of these records starts with the text:
`//First`

Next records General information about the topology.
If NELGRP > 1, NELGRP+1 records are used. The first record is a text record with the text
`//Element_Group_Information`
Of the next NELGRP records each record ielgrp refers to one element group in the natural sequence. The contents of these records is as follows:

Pos. 1: NPELM_ielgrp Number of nodal points in elements with element group sequence number ielgrp.

Pos. 2: ISHAPE_ielgrp Type number for mesh generation of elements with element group sequence number ielgrp.

Pos. 3: NELEM_ielgrp Number of elements with element group sequence number ielgrp.

Pos. 4: NBOUN_ielgrp Number of nodal points at the boundaries of elements with element group sequence number ielgrp.

Next record Text record with the text:
`//Coordinates`

Next record Co-ordinates of the mesh.
The co-ordinates are stored in the sequence:
$x_1, y_1, z_1, x_2, y_2, ..., z_{NPOINT}$
NDIM × NPOINT co-ordinates must be stored.

Next record Text record with the text:
`//Topology`

Next records Topology of the mesh. In this case for each element group ielgrp a new record is used. For each element group the nodal point numbers of all elements must be stored in the sequence: nodes of element 1, nodes of element 2 and so on. The sequence of the node numbers of the elements is essential and must correspond to Table 2.1.1 in the Users Manual.

Next record text record with the text:
`//User_Points`

Next record Nodal point sequence numbers of the user points.
This record is only used if NUSPNT > 0.
The nodal point sequence numbers of all user points must be stored sequentially. If nodal points numbers are ≤ 0, this means that no node number is connected to this user point.

Next records Information of the curves. These records are only used if NSURFS > 0.
In that case 2 × NCURVS+1 records are used. The first one is a text record with the text:
`//Curves`
The next group of two records each referring to one curve icurve.
For each curve, record 1 contains two integers:

1 Number of points at the curve.

2 Indication if the boundary is internal (0) or a boundary curve (1)

The second record contains the nodal point sequence numbers of the nodes at the curve, sequentially. If nodal points numbers are ≤ 0, this means that no node number is connected to this node at the curve.

Next records Information of the surfaces.
These records are only used if NVOLMS > 0.
In that case 2 × NSURFS+1 records are used. The first one is a text record with the text:
`//Surfaces`
The next group of two records each referring to one surface isurf.
For each surface, record 1 contains two integers:

1 Number of elements in the surface.

2 Number of nodes per element in the surface.

The second record contains the nodal point sequence numbers of the nodes in the sequence element 1, element 2 and so on. With respect to the sequence per element the sequence of Table 2.1.1 must be used.

Next records Possible renumbering of nodal points.
These records are only used if KELMJ > 0. The first one is a text record with the text:
`//Renumbering`.
The next one contains the new sequence of the nodes in correspondence with KMESH part j as described in Section 24.2.

Next record Text record with the text:
`//Kmesht`

Next record KMESH part t.
KMESH part t corresponds to the elements with the highest dimensionality, i.e. volume elements if NVOLMS>0, otherwise surface elements if NSURFS>0, otherwise curve elements. If NVOLMS > 0, then KMESH part t has length 2×NVOLMS, and KMESH t $(2 \times i - 1)$ contains the first element number of volume i, whereas KMESH t $(2 \times i)$ contains the last element number of volume i.
Otherwise if NSURFS > 0, then KMESH part t has length 2×NSURFS, and KMESH t $(2 \times i - 1)$ contains the first element number of surface i, whereas KMESH t $(2 \times i)$ contains the last element number of surface i.
Otherwise if NCURVS > 0, then KMESH part t has length 2×NCURVS, and KMESH t $(2 \times i - 1)$ contains the first element number of curve i, whereas KMESH t $(2 \times i)$ contains the last element number of curve i.

Next record KMESH part u.
This record is only used if KELMU>0. KMESH part u gives information with respect to the spectral elements. If KELMU>0, 5 values are stored which are exactly the same as described in KMESH part u in Section 24.2.
In that case the record is preceded by a text record containing the text
`//Spectral_Elements`
If version number > 1:

Next record text record with the text
`//Length_Info`

Next record 6 positions: lniinp, lnrinp, lenkmsh, lenkmsq, lenkmsr, lenkmss, defining the lengths of the next parts.

Next record text record with the text
`//Iinput`

Next record IINPUT
In this record the integer input as given by the user is stored. By writing this information to the file `meshoutput`, it is possible to create a mesh before starting a free surface program.
The length of this array is equal to lniinp.

Next record text record with the text
`//Rinput`

Next record RINPUT
In this record the real input as given by the user is stored.
The length of this array is equal to lnrinp.

Next record text record with the text
`//Kmeshq` (only if lenkmsq>0)

Next record KMESH part q.
This record is only written if KELMQ>0. If contains the subcurve numbers of each composite curve.
The length of this array is equal to lenkmsq.

Next record text record with the text
`//Kmeshr` (only if lenkmsr>0)

Next record KMESH part r.
This record is only written if KELMR>0. If contains the subsurface numbers of each composite surface.
The length of this array is equal to lenkmsr.

Next record text record with the text
`//Kmeshs` (only if lenkmss>0)

Next record KMESH part s.
This record is only written if KELMS>0. If contains the curve numbers corresponding to each surface.
The length of this array is equal to lenkmss.

Next record text record with the text
`//Kmeshh` (only if lenkmsh>0)

Next record KMESH part h.
This record is only written if KELMH>0 and IMAP=1. If contains information about the curves, the surfaces and the volumes.
The length of this array is equal to lenkmsh.

Next records Essential information of KMESH part h.
This record is only written if KELMH>0 and IMAP=1. If KMESH part h exists, some essential information is stored with respect to local curves, surfaces and volumes. The important parts of this information is stored in the file `meshoutput`. In that way array KMESH part h may be reconstructed in the reading program. This is an important issue in case of a changing mesh.
First the arrays icurvs, curves and curvefactors, as referred to in kmeshh(3), kmeshh(5) and kmeshh(6), (24.2) are written.
For each surface and volume a record of length $NPOINT_{local}$ is written, with $NPOINT_{local}$ the number of nodes in the surface or volume. This array contains the mapping of local numbering to global numbering.
Composite surfaces in R^3 are skipped.
Besides that for surfaces generated by general an extra record of this length is written containing very special internal information. In case of a composite surface an extra record of length NCURVS is written.

The following records are only used if IVERS≥4.

2 records with extra information.

Next record text record with the text
`//Dimension_Parameters`

Next record

Pos. 1: NLEVEL, i.e. the number of levels in the mesh. If 0, no levels are computed.

Pos. 2: NSPLIT, i.e. number of disjoint parts the mesh consists of. If the mesh contains only one part NSPLIT=0.

Pos. 3-10: 0 (Not yet used)

Next record

Pos. 1-10: 0 (Not yet used)

Next record text record with the text
`//Kmeshk`

Next record KMESH part k NLEVEL + NSPLIT positions containing KMESH part k as described in 24.2.

The following records are only used if IVERS≥5.

Next record text record with the text
`//Second_Dimension_Group`

Next record 10 positions.

Pos. 1: KMESH(45), i.e. indication if kmesh part v exists (KMESH(45)>0) or not (KMESH(45)=0).

Pos. 2: KMESH(46), i.e. indication if kmesh part w exists (KMESH(46)>0) or not (KMESH(46)=0).

Pos. 3: NUMOBST, i.e. number of obstacles.

Pos. 4: NEDGES, i.e. number of edges.

Pos. 5: NFACES, i.e. number of faces.

Pos. 6: Length of kmesh part x in integers.

Pos. 7: KMESH(58), i.e. indication if kmesh part AH (IOUTERBOUN) exists (KMESH(58)>0) or not (KMESH(58)=0).

Pos. 8: KMESH(61), i.e. indication if kmesh part AJ (NORMAL) exists (KMESH(61)>0) or not (KMESH(61)=0).

Pos. 9: KMESH(57), i.e. indication if kmesh part AG (VOLUME_SURFS) exists (KMESH(57)>0) or not (KMESH(57)=0).

Pos.10: KMESH(62), i.e. indication if kmesh part AK (CURVE_USERP) exists (KMESH(62)>0) or not (KMESH(62)=0).

The positions 4 and 5 are only filled if `write_faces` is given in the input.

Next record text record with the text
`//Kmeshv` (only if KMESH(45)>0)

Next records KMESH part v only if KMESH(45)>0.
The first record must contain lenkmsv, i.e. the length of KMESH part v. The following records contain the complete array KMESH part v as described in Section 24.2

Next record text record with the text
`//Kmeshw` (only if KMESH(46)>0)

Next records KMESH part w only if KMESH(46)>0.
The first record contains nintprop and nrealprop, i.e. the number of integer properties and real properties respectively.
This record must be followed by records containing the integer properties (nintprop*nelgrp positions) in the sequence all integer properties for element group 1, then for element group 2 and so on. Compare with Section 24.2.
These records must be followed by records containing the real properties (nrealprop*nelgrp positions) in the sequence all real properties for element group 1, then for element group 2 and so on. Compare with Section 24.2.

Next record text record with the text
`//Kmeshx` (only if NUMOBST>0)

Next record KMESH part x only if NUMOBST>0.
The first records must contain array kmeshx as described in Section 24.2.
The next records must contain array rmeshx as described in Section 24.2.
If NDIM=3, some extra records are used that contain the topology and coordinates of the obstacles. These records are not used when NDIM=2.
For each obstacle the following information is stored:
Record with the topology, i.e. inpelm × nelem local node numbers of the obstacle surface, followed by
Record with the coordinates of these nodes, in the sequence $x_1, y_1, z_1, x_2, ...$ Hence npoint ×

3 reals.
npoint is the number of points in the obstacle surface, nelem the number of elements in that surface and inpelm the number of nodes per element in that surface.

Next record text record with the text
`//Volume_surfs` (only if kmesh(57) >0)

Next record KMESH part AG only if kmesh(57)>0.
This record contains array `volume_surfs` as described in Section 24.2. This array contains all surfaces corresponding to the volumes.

Next record text record with the text
`//Outercurves` (only if kmesh(58) >0)

Next record KMESH part AH only if kmesh(58)>0.
This record contains array `outercurvs` as described in Section 24.2. This array contains all outer surfaces enclosing the volumes or if no volumes are present all outer curves enclosing the surfaces.

Next record text record with the text
`//Normals` (only if kmesh(61) >0)

Next record KMESH part AJ only if kmesh(61)>0.
This record contains array `normals` as described in Section 24.2. This array contains all normals defined on the curves (R^2) or surfaces (R^3).

Next record text record with the text
`//Curveusp` (only if kmesh(62) >0)

Next record KMESH part AK only if kmesh(62)>0.
This record contains array `CURVE_USERP` as described in Section 24.2. This array contains user point numbers corresponding to the curves.

The following records are only present from version 7:

Next records contains information of the edge numbers corresponding to the faces.
These records are only written if `write_faces` is used in the input. The first record is a text record with the text
`//Edges`
The next record contains the numbers nedges, nfaces.
The next record of length nfaces contains the starting positions of information of the edges as stored in NMESHB.
The last record contains the rest of NMESHB, i.e. for each face, the number of edges of the face followed by the edge sequence numbers.

Next records contains information of the faces corresponding to elements. These records are only written if `write_faces` is used in the input. The first record is a text record with the text
`//Faces`
The next record contains the numbers nelem, nfaces.
The next record of length nelem contains the starting positions of information of the edges as stored in NMESHA.
The last record contains the rest of NMESHA, i.e. for each element, the number of faces of the element followed by the face sequence numbers.

Next records contains information of the nodes corresponding to edges. These records are only written if `write_faces` is used in the input. The first record is a text record with the text
`//Nodes_per_edge`
The next record contains the numbers nedges.
The next record of length nedges contains the starting positions of information of the edges

as stored in NMESHC.
The last record contains the rest of NMESHC, i.e. for each edge, the number of nodes of the edge followed by the edge sequence numbers.

See Section 24.2 for a description of these arrays.

3.4 Reading the file meshoutput by subroutine MESHRD

Description

Subroutine MESHRD reads the file meshoutput as described in Section 3.3 and stores all information in array KMESH.

If subroutine MESHRD is used to read the mesh, SEPRAN may not be started with the starting subroutines SEPSTx, but these subroutines must be replaced by a series of calls. See Section 4.1 for the details.

Call

```
CALL MESHRD ( ICHOICE, IREF, KMESH )
```

Parameters

INTEGER ICHOICE, IREF, KMESH(∗)

KMESH In this array information of the mesh will be stored. This array has a variable length but in general a length of 100 positions is sufficient. The user must fill position 1 with the declared length. It is advised to set position 3 equal to 0.
For an exact definition of the contents of array KMESH the reader is referred to Section 24.2.

ICHOICE General choice parameter. Defines the actions with respect to the mesh. Possibilities:

-2 Standard case. The renumbering as stored in the file meshoutput is used in the program. All information that can be derived from the topology, like neighbours of elements and nodes is computed.

-1 Only the information of the mesh is read.
The renumbering stored in file meshoutput is ignored.
No information is derived from the topology.

≥ 0 The information of the mesh is read.
The renumbering stored in file meshoutput is ignored. In fact this option is meant for old versions of meshoutput only, where no renumbering of nodes has been stored.
Information is derived from the topology. The value of ICHOICE defines the renumbering algorithm to be applied.
ICHOICE = 0, means no renumbering.
ICHOICE > 0 means renumbering. The value of ICHOICE has exactly the same meaning as the parameter i in METHOD = i behind the keyword RENUMBER in the mesh input. See the Users Manual Section 2.2.

IREF Dataset reference number.
If IREF = 0, all input is read from the standard input file. This option is very unlikely.
If IREF > 0, all input is read (formatted) from the file with dataset reference number IREF. This file must have been opened before by the user.
If IREF < 0, all input is read (unformatted) from the file with dataset reference number -IREF. This file must have been opened before by the user.
In the standard case the parameter IREF is identical to the parameter IREF10 in common block CMACHN. See Section 21.17.
In order to open the standard file meshoutput subroutine INIFIL (Section 19.3) may be used.

Input

IREF and ICHOICE must have a value.
The file meshoutput must exist and have the structure as described in Section 3.3.

Output

Array KMESH has been filled with information of the mesh.

3.5 Writing the file meshoutput by subroutine MESHWR

Description

Subroutine MESHWR writes the mesh created by the mesh generator to the file meshoutput as described in Section 3.3.

Call

```
CALL MESHWR ( IREF, KMESH )
```

Parameters

INTEGER IREF, KMESH(∗)

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

IREF Dataset reference number.
If IREF = 0, all input is written to the standard input file. This option is very unlikely.
If IREF > 0, all input is written (formatted) to the file with dataset reference number IREF. This file must have been opened before by the user.
If IREF < 0, all input is written (unformatted) to the file with dataset reference number -IREF. This file must have been opened before by the user.
In the standard case the parameter IREF is identical to the parameter IREF10 in common block CMACHN. See Section 21.17.
In order to open the standard file meshoutput subroutine INIFIL (Section 19.3) may be used.

Input

IREF must have a value.
Array KMESH must have been filled.

Output

The mesh description is written to the file meshoutput as described in Section 3.3.

3.6 Refining an existing mesh uniformly (subroutine REFINE)

Description

Subroutine REFINE may be used to refine an existing mesh in such a way that each edge is subdivided into 2^n parts.

Call

```
CALL REFINE ( KMESH, ICHOICE, IINPUT )
```

Parameters

INTEGER KMESH(∗), ICHOICE, IINPUT(∗)

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

ICHOICE Choice parameter consisting of the parts NTIMES, ITOPOL and JRENUM according to the formula:

ICHOICE = NTIMES + 100×ITOPOL + 1000×JRENUM

These parameters have the following meaning:

NTIMES indicates the number of times the mesh must be refined. Hence each edge of the mesh is subdivided into 2^{NTIMES} parts.

ITOPOL indicates if extra information of the mesh that is necessary for the solution of problems must be stored in array KMESH. It concerns information about connections between elements, about neighbours and possibly a renumbering.
If ITOPOL=0 this extra information is computed and stored, if ITOPOL=1 it is not computed.
If the program is used to create a mesh that must be written by MESHWR (3.5 (Section 3.5), no renumbering is required and no further computations must be carried out by this program, ITOPOL may be chosen equal to one, otherwise ITOPOL=0 is necessary.

JRENUM indicates if nodal points must be renumbered and according to which method.
If JRENUM=0, the routine decides itself what type of renumbering must be used.
If JRENUM>0 the value of JRENUM has the same meaning as the parameter i in METHOD = i behind the keyword RENUMBER in the mesh input. See the Users Manual Section 2.2.
JRENUM=99 means no renumbering.
JRENUM is only used if ITOPOL=0.

IINPUT This array is only used when array KMESH is created by subroutine MESH. In that case it must be the result of a preceding call of subroutine MESH, with ICHOICE in the call of MESH ≥1.
When array KMESH is filled by subroutine MESHRD (3.4 then array IINPUT is not used.

Input

Array KMESH must have been filled.
ICHOICE must have a value.
When KMESH has been created by subroutine MESH, array IINPUT must have been filled.

Output

Array KMESH has been changed. The new mesh has been formed.
Figure 3.6.1 shows a mesh in original state and after 1 and 2 refinements.

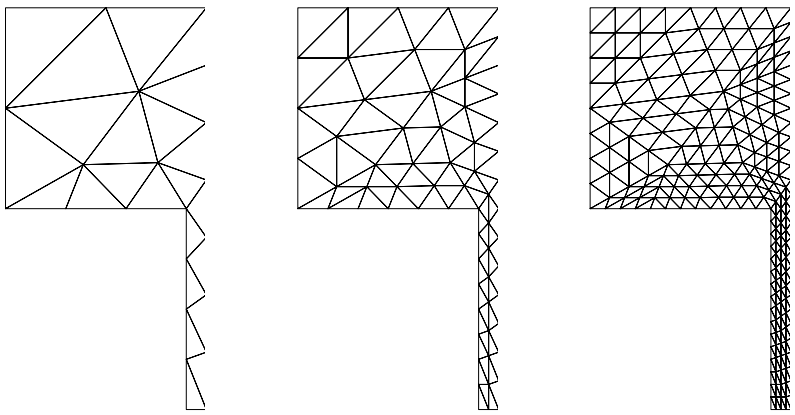


Figure 3.6.1: Mesh with 1 and 2 refinements.

3.7 Refining an existing mesh locally (subroutine REFLOC)

Description

Subroutine REFLOC may be used to refine an existing mesh locally. The user has to specify criterion values to indicate which parts have to be refined. This is only necessary for the first step of the process, if more refinements are needed, new values are adapted by REFLOC for the old points and are computed by interpolation for the new created points.
The contents of KMESH will be changed by this subroutine.

Call

```
CALL REFLOC ( KMESH, KPROB, JCRIT, USER, INTARR )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), JCRIT, INTARR(∗)

DOUBLE PRECISION USER(∗)

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

KPROB Standard SEPRAN array containing information about the problem description. KPROB must have been filled before for example by a SEPRAN start routine that reads the mesh.

JCRIT Criterion array, having the structure of a SEPRAN array that corresponds to an array with one unknown per point.
JCRIT corresponds to the double precision array CRITERION as described in INTARR.

USER Help array for the user to pass information to fill his own criterion array.
This array is only used if subroutine REFUSE is used, which depends on array INTARR.

INTARR Information array to describe which option of REFLOC has to be used. Array INTARR must be filled by the user in the following way:

Pos. 1 Number of positions that have been filled by the user in array INTARR. All positions that have not been filled will be treated as defaults. Hence INTARR(1) = 0, means that only default values will be used.

Pos. 2 Indicator whether a local criterion is given with the aid of array JCRIT (1) or is given in subroutine REFUSE (2). REFUSE is a simple user-written subroutine to fill the criterion array. See the example below. Default value : INTARR(2) = 2.

Pos. 3 Criterion indicator.

- if INTARR(3) = 1 a relative criterion will be used, which means that local refinement will go on as long as CRIT(i) > 1. After each refinement step the CRIT-values are decreased : CRIT(i) = CRIT(i) - 1
- if INTARR(3) = 2 an absolute criterion will be used, which means that local refinement will go on as long as the local length of the element sides is too long. Hence if length between nodes i and j is larger than (CRIT(i) + CRIT(j))/2 the side will be halved.
- Default value : INTARR(3) = 2.

Pos. 4 Maximal number of refine steps to be performed.
Default value : INTARR(4) = 10.

Pos. 5 JRENUM indicates if nodal points must be renumbered (JRENUM>0) or not (JRENUM=0). If JRENUM>0 the value of JRENUM has the same meaning as the parameter i in METHOD = i behind the keyword RENUMBER in the mesh input. See the Users Manual Section 2.2.
Default value: JRENUM=9.

Input

Array KMESH must have been filled.
Array KPROB must have been filled if INTARR(2) = 1
Array JCRIT must have been filled if INTARR(2) = 1
Array USER must have been filled if INTARR(2) = 2 and the user wants to use some special values later on in subroutine REFUSE.
At least INTARR(1) must have been filled.

Output

Array KMESH has been changed. The new mesh has been formed.

Subroutines called

If INTARR(2) = 1 or INTARR(1) = 0 REFLOC calls the user written subroutine REFUSE.

Subroutine REFUSE must be programmed as follows:

```
SUBROUTINE REFUSE( COOR, NPOINT, CRIT, USER, JREFIN )
INTEGER NPOINT, JREFIN
DOUBLE PRECISION COOR(*), CRIT(NPOINT), USER(*)

  statements to fill array CRIT

END
```

Meaning of the parameters:

COOR Co-ordinate array containing the coordinates sequentially. This array has been filled by subroutine REFLOC.

CRIT A criterion array which has to be filled by the user if JREFIN = 1 and may be filled if JREFIN > 1.

USER Is the array already described in REFLOC.

JREFIN Loop variable for actual refinement step. If JREFIN = 1 the refinement just started, hence there is not yet a criterion array given.

3.8 Transforming a mesh to other types of elements (subroutine TRANSF)

Description

Subroutine TRANSF transforms a mesh consisting of linear elements (lines, triangles and tetrahedrons) only into a mesh with other elements. The resulting mesh may contain quadratic triangles and tetrahedrons, bilinear quadrilaterals and hexahedrons or biquadratic quadrilaterals and hexahedrons.

Call

```
CALL TRANSF ( IINTRN, KMESH )
```

Parameters

INTEGER IINTRN(∗), KMESH(∗)

IINTRN Integer input array, in which some information about the process must be stored by the user.
IINTRN must be filled as follows:

Pos. 1 Highest entry number of IINTRN that has been filled by the user. The positions 1 to IINTRN(1) must have been given a value by the user. For all other positions default values are used.
Hence, IINTRN(1)=0 or 1 means that only defaults are used.

Pos.2 ITOPOL indicates if extra information of the mesh that is necessary for the solution of problems must be stored in array KMESH. It concerns information about connections between elements, about neighbors and possibly a renumbering. If ITOPOL=0 this extra information is computed and stored, if ITOPOL=1 it is not computed.
If the program is used to create a mesh that must be written by MESHWR (3.5 (Section 3.5), no renumbering is performed and no further computations must be carried out by this program, ITOPOL may be chosen equal to one, otherwise ITOPOL=0 is necessary.
Default value: ITOPOL=0.

Pos. 3 JRENUM indicates if nodal points must be renumbered (JRENUM>0) or not (JRENUM=0). If JRENUM>0 the value of JRENUM has the same meaning as the parameter i in METHOD = i behind the keyword RENUMBER in the mesh input. See the Users Manual Section 2.2.
JRENUM is only used if ITOPOL=0.
Default value: JRENUM=9.

Pos. 4 MESHTYPE indicates to what type of elements the linear triangles and tetrahedrons must be transformed. Possible values:

- 4 Transform each linear triangle into a quadratic triangle and each linear tetrahedron into a quadratic tetrahedron.
- 5 Transform each linear triangle into a bilinear quadrilateral and each linear tetrahedron into a trilinear hexahedron.
- 6 Transform each linear triangle into a biquadratic quadrilateral and each linear tetrahedron into a triquadratic hexahedron.
- 7 Transform each linear triangle into an extended quadratic triangle, i.e. a quadratic triangle with an extra point (7) in the center.

Default value: MESHTYPE=5.

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh. The contents of KMESH will be changed by this subroutine.

Input

Array KMESH must have been filled.
Array IINTRN must have been (partly) filled.

Output

A new mesh has been created. The contents of array KMESH have been changed.

Figures 3.8.1 and 3.8.2 show examples of meshes before and after transformation.

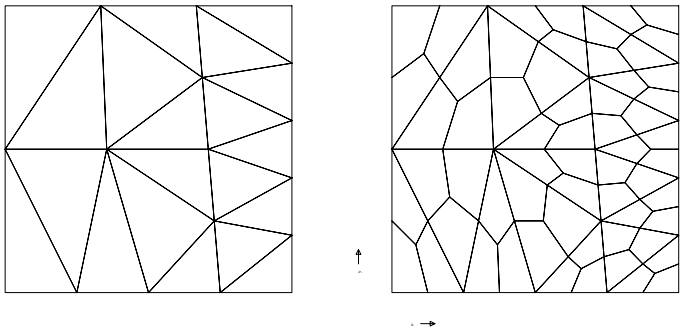


Figure 3.8.1: Example of a triangular mesh before and after transformation to a mesh consisting of quadrilaterals.

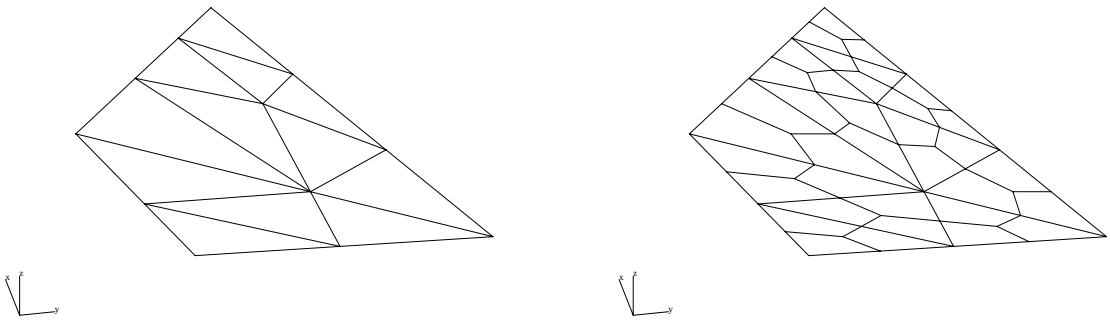


Figure 3.8.2: Example of a mesh consisting of tetrahedrons before and after transformation to a mesh consisting of hexahedrons

3.9 Adapting the boundary of a mesh

In the case of for example a free boundary problem, it may be necessary to change the mesh during the computations. In SEPRAN this action may be performed in two steps.
Assume that already some mesh has been created by subroutine MESH. It is not allowed to start with a mesh created by SEPMESH since the file *meshoutput* does not contain sufficient information.
The procedure is as follows:
First the boundary of the mesh is adapted by one of the subroutines CHANBN (3.9.3, ADAPBN (3.9.2 or ADAPBOUN (3.9.1.
Next the mesh is changed due to this new boundary. The new mesh may be topologically equivalent to the old one or may be completely new. Both possibilities may be carried out by subroutine MESH (3.2).

The following subroutines for adapting the boundary of the mesh are available:

CHANBN (3.9.3 is the oldest of the available boundary adaptation subroutines. It has fewer possibilities than the other subroutines and moreover it uses the arrays IINPUT and RINPUT explicitly. This means that these arrays must have been filled before, for example by a call to MESH with ICHOICE equal to 2.

ADAPBN (3.9.2 is more sophisticated. It does not use the arrays IINPUT and RINPUT and hence it is sufficient to start the computations with a call of MESH with ICHOICE equal to 0.

ADAPBOUN (3.9.1 is the most sophisticated one. ADAPBOUN (3.9.1 has the same possibilities as ADAPBN (3.9.2. However, input may not only be given by the parameter list but also through the standard input file. This may be done in each call of the subroutine but also in the first part of the input before the `end_of_sepran_input` record. In that case the starting subroutine SEPSTM (4.2.4 must have been used instead of the other ones.

We shall consider these subroutines separately.

3.9.1 Subroutine ADAPBOUN

Description

Subroutine to adapt the boundary of a region, where the mesh has been created by subroutine MESH, and will be recreated by subroutine MESH with the new boundary. ADAPBOUN is a typical subroutine for free-surface problems. At this moment ADAPBOUN is only available for 2D problems.

A call to ADAPBOUN must have been preceded by a call to MESH with ICHOICE any value, for example 0 or by a call to subroutine SEPSTM (4.2.4. It must be succeeded by a call to MESH with ICHOICE equal to 10 or 11.

Heading

```
subroutine adapboun ( inpada, rinada, kmesh, kprob, isol, icurvs, iread )
```

Parameters

INTEGER INPADA(∗), KMESH(∗), KPROB(∗), ISOL(5), ICURVS(∗), IREAD

DOUBLE PRECISION RINADA(∗)

INPADA Integer array to be filled by the user, indicating how the boundary must be adapted. INPADA is only used if IREAD_ACT = -1.
INPADA must be filled as follows:

Pos. 1 Position of last entry that has been filled by the user. If inpada(1) = 0 or 1, only defaults are used.

Pos. 2 NCURVES Number of curves that must be adapted. Default value: 1

Pos. 3 Not yet in use. At this moment a 0 must be supplied.
Default value: 0.

Pos. 4 IADAPT Indicates how the boundary must be adapted.
Possible values:

1 The curves are adapted by:
$\mathbf{x}_{new}(i) = \mathbf{x}_{old}(i) + \alpha(i)\mathbf{n}(i)$ with $\mathbf{x}_{old}$(i) the old co-ordinates of the i^{th} node and $\mathbf{x}_{new}$(i) the new ones.
$\mathbf{n}$(i) is the normal at the boundary of the i^{th} node and α(i) the $number^{th}$ degree of freedom in this node.
The normal is computed assuming a counter clockwise direction of the boundary.

2 The curves are adapted by the user provided subroutine FUNCCR as described in the Users Manual Section 3.3.9.

3 See 1, however, now α is computed according to the so-called 'film method' as described in Caswell and Viriyayuthakorn (1983).

$$\alpha(i) = \frac{factor\ (\psi(i) - \psi(0))}{|\mathbf{u} \cdot \mathbf{t}(i)|} \tag{3.9.1}$$

$$\psi(i) - \psi(0) = \int_{x(0)}^{x(i)} \mathbf{u} \cdot \mathbf{n}\ dx \tag{3.9.2}$$

The velocity $\mathbf{u}$ must be stored in the vector corresponding to isol. It is supposed that the degrees of freedom number and number+1 per point correspond to $\mathbf{u}$.
The flow rate in the first point of the curves to be adapted is assumed to be zero. This is used as start for the integration of the stream function ψ.
With respect to factor: see mult (INPADA(11)).

4 The curves are adapted by:
$\mathbf{x}_{new}(i) = \mathbf{x}_{old}(i) + factor\ \mathbf{u}(i)$ (Cartesian case) or
$(r,z)_{new}(i) = (r,z)_{old}(i) + \frac{factor}{2\pi}(u_r, u_z)(i)$ (Axisymmetric case)
With respect to factor: see mult (INPADA(11)).

5 See 1. Now α is computed according to $\alpha = factor\ \mathbf{u} \cdot \mathbf{n}$
With respect to factor: see mult (INPADA(11)).

6 The curves are adapted by:
$\mathbf{x}_{new}(i) = \mathbf{x}_{old}(i) + factor\ \mathbf{u} \cdot \mathbf{n}(i)$ with $\mathbf{x}_{old}$(i) the old co-ordinates of the i^{th} node and $\mathbf{x}_{new}$(i) the new ones.
$\mathbf{n}$(i) is the normal at the boundary of the i^{th} node.
$\mathbf{u}(i)$ is the velocity in point i.
With respect to $factor$, see mult (INPADA(11)).
In fact this is the same as IADAPT = 5, however, there is an essential difference.
In this particular case the velocity is assumed to be stored in the array corresponding to ISOL as a vector of special structure defined per elements, hence it is discontinuous. The velocity in the centroid of the elements at the curves are multiplied by the local normal and $factor$. In this way the displacement of the boundary in all mid points of the curve elements is defined. In order to compute the displacement in the vertices of the curve elements we proceed as follows:
The tangential vector $\mathbf{t}$ in vertex i is defined by: $\mathbf{t} = \mathbf{x}_{i+1} - \mathbf{x}_{i-1}$. The

normal $\mathbf{n}$ is perpendicular to this tangent vector, assuming a counter clockwise direction of the boundary.
Let the displacement in the two adjacent midpoints, as defined above, be $\delta\mathbf{x}_{i-\frac{1}{2}}$ and $\delta\mathbf{x}_{i+\frac{1}{2}}$. Hence the new positions of these points would be $\mathbf{x}^{new}_{i-\frac{1}{2}} = \mathbf{x}^{old}_{i-\frac{1}{2}} + \delta\mathbf{x}_{i-\frac{1}{2}}$ and $\mathbf{x}^{new}_{i+\frac{1}{2}} = \mathbf{x}^{old}_{i+\frac{1}{2}} + \delta\mathbf{x}_{i+\frac{1}{2}}$.
The displacement in the vertex is defined as $\delta\mathbf{x}_i = \alpha\mathbf{n}$, hence the new position of the vertex becomes: $\mathbf{x}^{new}_i = \mathbf{x}^{old}_i + \delta\mathbf{x}_i$. Now α is computed such that the area of the two quadrilaterals defined by $\mathbf{x}^{new}_i$, $\mathbf{x}^{new}_{i-\frac{1}{2}}$, $\mathbf{x}^{old}_{i-\frac{1}{2}}$, $\mathbf{x}^{old}_i$ and $\mathbf{x}^{new}_i$, $\mathbf{x}^{new}_{i+\frac{1}{2}}$, $\mathbf{x}^{old}_{i+\frac{1}{2}}$, $\mathbf{x}^{old}_i$ is equal to the mean value of the rectangles defined by the length of the adjacent curve elements multiplied by their respective midpoint displacement.
This definition of the displacement in the vertices can not be applied for the two end points of the curves. For these two points we use the displacement computed in the midside points.
The reason for using this quite complicated definition is that it provides a kind of smoothing compared to IADAPT = 5. This smoothing is such that if the displacement in the midpoints is equal to zero, also the displacement in the vertices is equal to zero.
At this moment IADAPT=6 is only implemented for linear elements.

7 The curves are adapted as in described for iadapt = 6, however, the more accurate scheme described in Segal, Vuik and Vermolen is used Use omega, eps and maxiter as stored in rinada(7), rinada(6) and inpada(18). See the Users Manual Section 3.4.4.

8 The curves are adapted by:
$\mathbf{x}_{new}(i) = \mathbf{x}_{old}(i) + factor\ \mathbf{u}$, where $\mathbf{u}$ is a constant velocity.
The value of the velocity is given in rinada(15) to rinada(17).
Factor must be stored in rinada(1).

9 The curves are adapted by the user provided subroutine FUNCSOLCR as described in the Users Manual Section 3.3.15.

Default value: 1

Pos. 5 IDIREC Indicates the direction of the computation If idirec = 1 the computation is performed from first to last point, if idirec = -1 from last to first point. It is sometimes required to alternate between 1 and -1 during iteration.
This option is only used for iadapt = 3
Default value: 1

Pos. 6 NUMBER the $number^{th}$ degree of freedom of isol (and higher) are used for the computation of α. See iadapt.
Default value: 1

Pos. 7 LINEAR Indicates if the elements along the boundary must be treated as linear or as quadratic elements.
Possible values:

1 linear

2 quadratic

Default value: to be computed by the subroutine in the following way: If all internal elements are quadratic, the boundary elements are assumed quadratic, otherwise linear.

Pos. 8 JCART defines the type of co-ordinate system to be used. This is of importance for iadapt = 3, 4 and 5.
Possible values:

1 Cartesian co-ordinates

2 Axi-symmetric co-ordinates

Default value: 1

Pos. 9 IEXLUDE_1: indicates which of the co-ordinates of the first point of the first curve must be adapted.
Possible values:
- 0 Both co-ordinates must be adapted
- 1 Only the second co-ordinate must be adapted, the first one remains unchanged
- 2 Only the first co-ordinate must be adapted, the second one remains unchanged
- 3 Both co-ordinates remain unchanged

Default value: 0

Pos. 10 IEXLUDE_2: indicates which of the co-ordinates of the last point of the last curve must be adapted.
Possible values:
- 0 Both co-ordinates must be adapted
- 1 Only the second co-ordinate must be adapted, the first one remains unchanged
- 2 Only the first co-ordinate must be adapted, the second one remains unchanged
- 3 Both co-ordinates remain unchanged

Default value: 0

Pos. 11 MULT indicates how the value of factor must be computed. (iadapt>2)
Possible values:
- 0 Multiplication factor is 1
- 1 Multiplication factor is the parameter dt from common ctimen
- 2 Multiplication factor is stored in rinada(1)
- 3 Multiplication factor is dt $\times$ rinada(1)

Default value: 0

Pos. 12 ITHRES indicates if a threshold value is used in Caswell's method and where to find it.
If $|\mathbf{u} \cdot \mathbf{t}(i)|$ larger than the threshold value the boundary is adapted, otherwise the old co-ordinates are used.
Possible values:
- 0 Threshold value = machine accuracy (usually 10^{-15})
- 1 Threshold value is stored in rinada(2)

Default value: 0

Pos. 13 IANGLE_1 Indicates if angle of free surface at first point is prescribed or not.
Possible values:
- 0 angle is not given
- 1 angle is stored in rinada(3)
- <0 The first point of the curves must be projected onto the curve $c_{|iangle_1|}$

Default value: 0

Pos. 14 IANGLE_2: Indicates if angle of free surface at last point is prescribed or not.
Possible values:
- 0 angle is not given
- 1 angle is stored in rinada(4)
- <0 The last point of the curves must be projected onto the curve $c_{|iangle_2|}$

Default value: 0

Pos. 15 IREDISTRIBUTE Indicates if the nodes at the newly computed boundary must be redistributed or not.
IREDISTRIBUTE consists of two parts iredist_type and ncurves_redist according to: IREDISTRIBUTE = iredist_type + 100 $\times$ ncurves_redist.
Possible values for iredist_type

0 No redistribution takes place, i.e. the nodes get the newly computed coordinates.

1 Once the new boundary is computed, this boundary is approximated by a spline and the nodes along this new boundary are redistributed according to the coarseness defined in the user points.

2 See 1, however, in this case the number of nodes may also be adapted. Hence the number of nodes at the boundary may have been decreased or increased after the distribution.
Mark that in this case the topology of the mesh is always changed.

ncurves_redist defines the number of curves that must be redistributed according to iredist_type.
If ncurves_redist = 0, all curves are redistributed according to iredist_type.
If ncurves_type > 0, the curve numbers of the curves that must be redistributed must be stored in array ICURVS positions NCURVES+1 to NCURVES+ncurves_redist.
If redistribution takes place, the curve is approximated by a spline and the factor α as described in the Users Manual Section 2.3 must be stored in rinada(5).
Default value: 0

Pos. 16 IPLOTBOUNDARY Indicates if the new boundary must be plotted or not.
Possible values:

0 No plotting.

1 Make a plot of the boundary

2 plot new boundary, use xmin, xmax, ymin and ymax from rinada(8-11)

3 plot new boundary, use yfact from rinada(14)

4 plot new boundary, use xmin, xmax, ymin and ymax from rinada(8-11) and yfact from rinada(14)

Default value: 0

Pos. 17 NUMOBST Defines the number of obstacles.
Each obstacle must be closed and consist of one curve only (This may be a curve of curves). At most 10 obstacles may be used. The curve numbers of the Numobst obstacles are stored in array icurvs positions: ncurves+ncurves_redist+1 ... .+Numobst
Default value: 0

Pos. 18 MAXITER, defines the maximum number of iterations in the iteration process.
Default value: 10

Pos. 19 PRIN_BOUN, defines whether the new boundary must be plotted or printed.
Possible values:

0 do not plot new boundary.

1 print new boundary.

2 print new boundary, use xmin, xmax, ymin and ymax from rinada(8-11).

Default value: 10

Pos. 20 CHECK_DIR makes only sense in combination with IADAPT = 7. If this keyword is used it is checked if the update of the boundary is in the direction of the outward or inward pointed normal depending on the value of CHECK_DIR.
Possible values:

-1 It is check if the change of the boundary is in the same direction as the inwards pointed normal.

0 No check is carried out.

1 It is check if the change of the boundary is in the same direction as the outwards pointed normal.

Default value: 0

RINADA Real input array with data that must be filled by the user as described in the input of INPADA. It is recommended to make RINADA large enough and

to initialize all positions that are not used by 0. At this moment a length of 50 positions is sufficient.

KMESH Array containing information about the mesh to be changed.
Array KMESH must have been filled by subroutine MESH (3.2) and may not have been filled by subroutine MESHWR (3.5.
A part of the arrays corresponding to KMESH as stored in the buffer array IBUFFR will be changed by ADAPBOUN.

KPROB Array containing information about the problem definition.
Array KPROB must have been filled before.

ISOL Array containing information about the solution vector. This vector is used to update the boundaries, see INPADA.

ICURVS Integer array containing the curve numbers for the adaptation of the boundary.
Array ICURVS consists of three parts.

Part a contains all curve numbers of the curves that must be adapted. This part has length NCURVES.
A negative curve number indicates that the curve must be computed in the reversed direction.
The curves including their sign must form one contiguous curve.

Part b is only used if ncurves_redist>0. It has length ncurves_redist and it must contain the curve numbers of the curves that may be redistributed.

Part c is only used if numobst>0. It has length numobst it must contain the curve numbers corresponding to the obstacles.

The number of curves is stored in array inpada.
icurvs is only used if iread_act = -1.

IREAD This parameter consists of two parts IREAD_ACT and NCALL according to: iread = sign(iread_act)*(iread_act+1000*times*ncall) with

iread_act Defines if parameters should be read or are given in array INPADA through the parameter list.
Possible values:

- 0 The input is read from the standard input file.
- > 0 The input as read from the standard input file by subroutine SEPSTM (4.2.4 is used. The value of iread defines the sequence number.
- -1 No input is read, all information from array INPADA is used.

For a description of the input through the standard input file the user is referred to the Users Manual Section 3.4.4.

ncall Number of subsequent calls to adapboun. If zero 1 is assumed.
Remark: only after the last of the NCALL calls the updating of the boundary is completed.

Input

IREAD must have a value.
The arrays KMESH, KPROB and ISOL must have been filled.
Depending on the value of IREAD the arrays INPADA, RINADA and ICURVS must have been filled.

Output

Some arrays stored in IBUFFR and corresponding to KMESH have been changed.

3.9.2 Subroutine ADAPBN

Description

Subroutine to adapt the boundary of a region, where the mesh has been created by subroutine MESH, and will be recreated by subroutine MESH with the new boundary. ADAPBN is a typical subroutine for free-surface problems. At this moment ADAPBN is only available for 2D problems.

A call to ADAPBN must have been preceded by a call to MESH with ICHOICE any value, for example 0. It must be succeeded by a call to MESH with ICHOICE equal to 10 or 11.

Heading

```
subroutine adapbn ( inpada, rinada, kmesh, kprob, isol )
```

Parameters

INTEGER INPADA(∗), KMESH(∗), KPROB(∗), ISOL(5)

DOUBLE PRECISION RINADA(∗)

INPADA Integer array to be filled by the user, indicating how the boundary must be adapted. INPADA must be filled as follows:

Pos. 1 Position of last entry that has been filled by the user. If inpada(1) = 0 or 1, only defaults are used.

Pos. 2 ICRV1 Curve number of first curve to be adapted
Default value: 1

Pos. 3 ICRV2 Curve number of last curve to be adapted.
Remark: the curve numbers icrv1 to icrv2 must be subsequent curves. In case a series of adjacent curves must be adapted, which are not subsequent curves it is advised to create a curve consisting of subcurves and use this new curve. If curves to be adapted are not adjacent, several calls of ADAPBN may be necessary.
Default value: icrv1.

Pos. 4-15 See subroutine ADAPBOUN (3.9.1 3.9.1. Those options that require array ICURVS have obviously not been implemented.

RINADA,KMESH,KPROB,ISOL See subroutine ADAPBOUN (3.9.1.

Input

The arrays INPADA, RINADA, KMESH, KPROB and ISOL must have been filled.

Output

Some arrays stored in IBUFFR and corresponding to KMESH have been changed.

3.9.3 Subroutine CHANBN

Description

Subroutine to adapt the boundary of a region, where the mesh has been created by subroutine MESH, and will be recreated by subroutine MESH with the new boundary. CHANBN is a typical subroutine for free-surface problems. At this moment CHANBN is only available for 2D problems.

A call to CHANBN must have been preceded by a call to MESH with ICHOICE equal to 1 to 5, i.e. the arrays IINPUT and RINPUT must have been filled by MESH or correspond to MESH.
It must be succeeded by a call to MESH with ICHOICE equal to 4 or 5.

Heading

```
subroutine chanbn ( ichoicha, number, icrv1, icrv2, kmesh, kprob, isol, iinput, rinput,
                    iuser, user )
```

Parameters

INTEGER ICHOICHA, NUMBER, ICRV1, ICRV2, KMESH(∗), KPROB(∗), ISOL(5), IINPUT(∗), IUSER(∗)

DOUBLE PRECISION RINPUT(∗), USER(∗)

ICHOICHA Choice parameter consisting of three parts CHOICE1, CHOICE2 and CHOICE3 according to ICHOICHA = CHOICE1 + CHOICE2 + CHOICE3. With

CHOICE1 Choice parameter indicating how the boundary must be updated. Possible values:

- 1 See subroutine ADAPBN (3.9.2 (IADAPT=1 and LINEAR=1)
- 2 See subroutine ADAPBN (3.9.2 (IADAPT=1 and LINEAR=2)
- -1 See subroutine ADAPBN (3.9.2 (IADAPT=2 and LINEAR=1)
- -2 See subroutine ADAPBN (3.9.2 (IADAPT=2 and LINEAR=2)
- 3 See subroutine ADAPBN (3.9.2 (IADAPT=3, LINEAR=1, IDIREC=1)
- 4 See subroutine ADAPBN (3.9.2 (IADAPT=3, LINEAR=2, IDIREC=1)
- -3 See subroutine ADAPBN (3.9.2 (IADAPT=3, LINEAR=1, IDIREC=-1)
- -4 See subroutine ADAPBN (3.9.2 (IADAPT=3, LINEAR=2, IDIREC=-1)
- 5 See subroutine ADAPBN (3.9.2 (IADAPT=3, LINEAR=1, IDIREC=1, JCART=2)
- 6 See subroutine ADAPBN (3.9.2 (IADAPT=3, LINEAR=2, IDIREC=1, JCART=2)
- -5 See subroutine ADAPBN (3.9.2 (IADAPT=3, LINEAR=1, IDIREC=-1, JCART=2)
- -6 See subroutine ADAPBN (3.9.2 (IADAPT=3, LINEAR=2, IDIREC=-1, JCART=2)
- 7 See subroutine ADAPBN (3.9.2 (IADAPT=4, LINEAR=1, MULT=2)
- 8 See subroutine ADAPBN (3.9.2 (IADAPT=4, LINEAR=2, MULT=2)
- -7 See subroutine ADAPBN (3.9.2 (IADAPT=4, IDIREC=-1, LINEAR=1, MULT=2)
- -8 See subroutine ADAPBN (3.9.2 (IADAPT=4, IDIREC=-1, LINEAR=2, MULT=2)
- 9 See subroutine ADAPBN (3.9.2 (IADAPT=4, LINEAR=1, MULT=2, JCART=2)
- 10 See subroutine ADAPBN (3.9.2 (IADAPT=4, LINEAR=2, MULT=2, JCART=2)

-9 See subroutine ADAPBN (3.9.2 (IADAPT=4, IDIREC=-1, LINEAR=1, MULT=2, JCART=2)

-10 See subroutine ADAPBN (3.9.2 (IADAPT=4, IDIREC=-1, LINEAR=2, MULT=2, JCART=2)

11 See subroutine ADAPBN (3.9.2 (IADAPT=4, LINEAR=1, MULT=3)

12 See subroutine ADAPBN (3.9.2 (IADAPT=4, LINEAR=2, MULT=3)

-11 See subroutine ADAPBN (3.9.2 (IADAPT=4, IDIREC=-1, LINEAR=1, MULT=3)

-12 See subroutine ADAPBN (3.9.2 (IADAPT=4, IDIREC=-1, LINEAR=2, MULT=3)

CHOICE2 Indicates which of the co-ordinates of the first point of the first curve must be adapted.
Possible values:

0 Both co-ordinates must be adapted

100 Only the second co-ordinate must be adapted, the first one remains unchanged

200 Only the first co-ordinate must be adapted, the second one remains unchanged

300 Both co-ordinates remain unchanged

CHOICE3 Indicates which of the co-ordinates of the last point of the second curve must be adapted.
Possible values:

0 Both co-ordinates must be adapted

1000 Only the second co-ordinate must be adapted, the first one remains unchanged

2000 Only the first co-ordinate must be adapted, the second one remains unchanged

3000 Both co-ordinates remain unchanged

NUMBER See subroutine ADAPBN (3.9.2.

ICRV1 See subroutine ADAPBN (3.9.2.

ICRV2 See subroutine ADAPBN (3.9.2.

KMESH See subroutine ADAPBN (3.9.2.

KPROB See subroutine ADAPBN (3.9.2.

ISOL See subroutine ADAPBN (3.9.2.

IINPUT This array must have been filled by subroutine MESH. It contains a part of the description of the mesh.

RINPUT This array must have been filled by subroutine MESH. It contains the rest of the description of the mesh.

IUSER will not be used.

USER will not be used.

Input

The arrays IINPUT, RINPUT, KMESH, KPROB and ISOL must have been filled.
The parameters ICHOICHA, NUMBER, ICRV1, ICRV2 must have a value.

Output

Some arrays stored in IBUFFR and corresponding to KMESH have been changed.

3.10 Adapting a mesh for a free boundary problem (subroutine ADAPMESH)

Description

In the case of for example a free boundary problem, it may be necessary to change the mesh during the computations. Subroutine ADAPMESH (3.10 performs this task by changing the boundary first and afterwards the mesh.
In each step of the iteration or in each time-step a solution is computed. This solution implicitly defines a new boundary for the free boundary problem, and hence the mesh must be adapted to this new boundary.
Adaptation of boundary and mesh is performed by the subroutines ADAPMESH (3.10 and ADAPBOUN (3.9.1. ADAPBOUN itself is called by ADAPMESH (3.10.
Adapting the boundary means that the mesh is changed but not necessarily the topology. Sometimes it is sufficient to update the co-ordinates of the mesh. This is much more cheaper than remeshing of the entire mesh. However, changing the co-ordinates only may result in very distorted elements, in which case remeshing must be applied. Whether or not the topology is adapted is the responsibility of the user.

Call

```
CALL ADAPMESH ( KMESH, KPROB, INTMAT, ISOL, INPADA, RINADA,
                INPMSH, RINMSH, ICURVS, IREAD )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), INTMAT(5), ISOL(5), INPADA(∗), INPMSH(∗), ICURVS(∗), IREAD

DOUBLE PRECISION RINADA(∗), RINMSH(∗)

KMESH Array containing information about the mesh to be changed.
Array KMESH must have been filled by subroutine MESH (3.2), or subroutine SEPSTM (4.2.4 (4.2) and may not have been filled by subroutine MESHWR (3.5 (3.5).
A part of the arrays corresponding to KMESH as stored in the buffer array IBUFFR will be changed by ADAPMESH (3.10.

KPROB Array containing information about the problem definition.
Array KPROB must have been filled before.
If the topology of the mesh is changed also KPROB will be adapted.

INTMAT Array containing information about the structure of the large matrix.
Array INTMAT must have been filled before.
If the topology of the mesh is changed also INTMAT will be adapted.

ISOL Array containing information about the solution vector. This vector is used to update the boundaries, see INPADA.

INPADA Integer array to be filled by the user, indicating how the boundary must be adapted.
INPADA is only used if iread = -1. INPADA contains the input arrays INPADA for the various calls of adapboun sequentially.
Hence:
INPADA(1) contains the last entry for call 1 (ilast1).
INPADA(1) to INPADA(ilast1) contains the complete array INPADA for the first call of adapboun.
INPADA(1+ilast1) contains the last entry for call 2 (ilast2), however, shifted with ilast1.
So the actual last entry is ilast1+ilast2.
INPADA(ilast1+1) to INPADA(ilast1+ilast2) contains the complete array INPADA for the second call of adapboun.

This process is repeated.

The easiest way to implement this in the calling program is to define a more dimensional array INPADA(LENADA,NCALL), and to give INPADA(1,i) the value LENADA

RINADA Real input array with data that must be filled by the user as described in the input of INPADA.

INPMSH Integer array to be filled by the user, indicating how the mesh must be adapted.
INPMSH is only used if iread = -1.
INPMSH must be filled as follows:

1 Position of last entry that has been filled by the user. If inpmsh(1) = 0 or 1, only defaults are used.

2 NCALL: Number of calls of adapboun that must be used to update the complete boundary before updating the mesh.
In this way it is possible to update parts of the boundary separately.
Default value: 1

3 CHANGE_TOPOLOGY: defines whether the topology must be adapted or not.
Possible values:

- 0 The topology remains unchanged, hence only the co-ordinates of the mesh are changed.
- 1 The topology is always adapted. This means that remeshing is applied. Not only the mesh is changed also the problem definition as stored in KPROB and the matrix structure as stored in INTMAT are adapted.
- 2 The topology is adapted if necessary.
  This possibility has not yet been implemented.
- 3 The adaptation of the topology is performed in two steps.
  First the mesh is updated without changing the topology.
  Then the arrays kmesh and kprob are copied for interpolation purposes.
  Next remeshing is applied.
  This option is an extension of 1 and makes only sense in combination with inpmsh(5)>0.

Default value: 0

4 PLOT_MESH: defines whether the new mesh must be plotted or not.
Possible values:

- 0 No plot is made.
- 1 The mesh is plotted.

Default value: 0

5 INTERPOLATE_SOL: defines the number of solution vectors that must be interpolated.
Default value: 0

6 INDEX_SOL: defines the starting address in INPMSH of the vectors to be interpolated.
In positions INPMSH(INDEX_SOL) to INPMSH(INDEX_SOL+INTERPOLATE_SOL-1) the sequence numbers of the solution vectors to be interpolated are stored.
If INDEX_SOL = 0, the vectors 1 to INTERPOLATE_SOL are interpolated.
If INDEX_SOL < 0, the vectors -INDEX_SOL to -INDEX_SOL+INTERPOLATE_SOL-1 are interpolated.
Default value: 0

RINMSH See INPMSH.

ICURVS Integer array containing the curve numbers for the adaptation of the boundary.
A negative curve number indicates that the curve must be computed in the reversed direction.
The curves including their sign must form one contiguous curve.
The number of curves is stored in array inpada.

icurvs is only used if iread = -1.
If the boundary must be adapted in a series of calls, i.e. NCALL>1, then the curves must be stored in the sequence of the various calls. Hence first all ncurves curves for the first call, then for the second one and so on.

IREAD Defines if parameters should be read or are given in the arrays INPMSH and INPADA through the parameter list.
Possible values:

- 0 The input is read from the standard input file.
- > 0 The input as read from the standard input file by subroutine SEPSTM (4.2.4 is used. The value of iread defines the sequence number with respect to the keyword ADAPT_MESH.
- -1 No input is read, all information from the arrays INPADA and INPMSH is used.

For a description of the input through the standard input file the user is referred to the Users Manual Section 3.4.3.

Input

IREAD must have a value.
The arrays KMESH, KPROB, INTMAT and ISOL must have been filled.
Depending on the value of IREAD the arrays INPMSH, RINMSH, INPADA, RINADA and ICURVS must have been filled.

Output

Some arrays stored in IBUFFR and corresponding to KMESH have been changed.
The arrays KPROB and INTMAT and their corresponding arrays in IBUFFR have been changed if the topology is adapted.

3.11 Creating a linear finite element mesh from a spectral mesh (subroutine FEMESH)

Description

Subroutine FEMESH may be used to create a linear finite element mesh from an existing spectral element mesh. The finite element mesh consists of a mesh of (bi/tri) linear elements, based on the nodal points of the spectral mesh. Hence the number of nodes in both meshes are equal, but the number of elements in the finite element mesh is considerably larger than that of the spectral mesh. This routine is used to create a finite mesh which can be used to display the solution. This routine may also be used to create the finite element preconditioner to be used in iterative methods.

Heading

```
subroutine femesh ( kmesh1, kmesh2, iref, iren )
```

Parameters

INTEGER KMESH1(∗), KMESH2(∗), IREF, IREN

KMESH1 Standard SEPRAN array containing information about the spectral element mesh.

KMESH2 Standard SEPRAN array containing information about the finite element mesh to be created.

IREF Device number for the finite element meshoutput file. This file is called `femesh.out` and may be used for postprocessing purposes.

IREN Parameter to indicate if the finite element mesh must be renumbered (IREN = 1) or not (IREN = 0).

Input

Array KMESH1 must have been filled.
The device number IREF and renumbering parameter IREN must have a value.

Output

Array KMESH2 has been filled with the linear finite element mesh.
If IREN = 1, the mesh is renumbered and also the numbering of the spectral element mesh KMESH1 is changed according to the numbering of KMESH1.
Information of the mesh is written to the file `femesh.out`.

3.12 Deforming a mesh due to displacements. (subroutine DEFMSH)

Description

Subroutine DEFMESH is used to deform a mesh due to already computed deformations. In fact the co-ordinates of the mesh are recomputed given a displacement in the solution vector.
The old co-ordinates are destroyed.

This subroutine is usually applied for the following reasons

- To plot deformed 3D meshes
- To deform meshes for Arbitrary Euler Lagrange computations
- To deform meshes for Updated Lagrange computations

Heading

```
subroutine defmsh ( kmesh, kprob, isol, scale )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), ISOL(∗), SCALE

KMESH Standard SEPRAN array containing information about the mesh to be deformed.

KPROB Standard SEPRAN array containing information about the problem definition.

ISOL Solution array or array of special structure containing the displacement.

SCALE scaling factor. The displacement is multiplied by SCALE. A value unequal to 1 is usually applied if the routine is applied to plot the deformed mesh and if the deformations are small.

Input

The arrays KMESH, KPROB and ISOL must have been filled.
The parameter SCALE must have a value (usually 1).

Output

Array KMESH has been changed. In fact only the co-ordinates have been updated.

4 Start subroutines and problem definition

4.1 Introduction

In this chapter the various SEPRAN starting subroutines, subroutines to read the problem definition and subroutines to define the structure of the matrix are treated.
In fact SEPRAN contains one general subroutine SEPSTR 4.2.1 that performs all these tasks. This subroutine itself consists of a number of subroutines performing sub tasks:

START (4.3) actual SEPRAN starting subroutine

MESHRD (3.4) reads the mesh generated by SEPMESH

PROBDF (4.4) reads the problem definition

MATSTRUC (4.5.1) defines the structure of the matrix

If the mesh has not been created before by program SEPMESH, but the user calls subroutine MESH (3.2) instead, it is necessary to use these subroutines separately. In that case the call of SEPSTR (4.2.1) in the example in Section 2.1 (Reference 6) must be replaced by

```
istart = ...
irotat = ...
ioutp  = ...
itime  = ...
call start ( istart, irotat, ioutp, itime )
ichoice = ...
call mesh ( ichoice, iinput, rinput, kmesh )
ichoiceprob = ...
call probdf ( ichoiceprob, kprob, kmesh, iinputprob )
iincommt(1) = ...
iincommt(2) = ...
    .
    .
    .
call matstruc ( iincommt, kmesh, kprob, intmat )
```

The call to MESH may also be replaced by a call to MESHRD (3.4) in case the mesh must be read from the file meshoutput.

Besides SEPSTR (4.2.1) SEPRAN has some alternative subroutines: SEPSTL (4.2.2), SEPSTN (4.2.3) and SEPSTM (4.2.4.
These subroutines have exactly the same task as SEPSTR (4.2.1), but provide some extra possibilities. SEPSTL (4.2.2) for example, allows the user to give the length of array IBUFFR as parameter, which makes it more easily to extend array IBUFFR.
SEPSTN (4.2.3 is an extension of these subroutines. Not only performs it the same tasks as subroutine SEPSTL (4.2.2), besides that, it allows the program to read all SEPRAN input until the record

```
end_of_sepran_input
```

is found. If SEPSTN (4.2.3) is not used all subroutines read the input as soon as they are called. This means that the sequence of the input in the input file is not longer more position independent. Moreover, it may be possible that private input of the user must be put between the rest of the SEPRAN input.
Another extension of SEPSTN (4.2.3) is formed by SEPSTM (4.2.4), which does not read the mesh

from the file `meshoutput` but creates the mesh by reading user input. SEPSTM (4.2.4) is meant for free surface problems.

In this chapter we deal with the following paragraphs:

4.2 treats the starting subroutines SEPSTR (4.2.1), SEPSTL (4.2.2), SEPSTN (4.2.3) and SEPSTM (4.2.4).

4.3 describes the simple starting subroutine START (4.3).

4.4 gives a description of subroutine PROBDF (4.4) that reads and interprets the problem definition.

4.5 deals with subroutine MATSTRUC (4.5) that is used to define the structure of the matrix.

4.6 treats subroutine CHTYPE (4.6), which may be used to change the type numbers of the elements from one series to another one.

4.7 treats subroutine FILLINP (4.7), which may be used to store local input arrays into the buffer in the same way standard input by sepcomp is stored.

4.2 The general starting subroutines SEPSTR, SEPSTL, SEPSTN and SEPSTM

In this section the available general starting subroutines will be treated. The standard general starting subroutine is SEPSTR (4.2.1). This subroutine starts SEPRAN, initializes common blocks, reads the problem definition and the information about the structure of the matrices as well as the mesh from the file meshoutput.
If the user wants to extend array IBUFFR as described in Section 1.2, he could use subroutine SEPSTL (4.2.2), since this subroutine has the declared length of IBUFFR as input.
The most general of the starting subroutines is subroutine SEPSTN (4.2.3), which performs the same tasks as SEPSTL (4.2.2), but has also the possibility to read and store all SEPRAN data from the input file until the record

```
end_of_sepran_input
```

is found.

A special starting subroutine is subroutine SEPSTM (4.2.4) which has exactly the same tasks as SEPSTN (4.2.3), however, instead of reading the file meshoutput this subroutine creates the mesh itself and therefore requires input for the mesh generation part. Of course the use of SEPSTM (4.2.4) increases the program size considerably. SEPSTM (4.2.4) is meant for free and moving boundary problems.

4.2.1 Subroutine SEPSTR

Description

SEPSTR starts the SEPRAN program, which means that it initializes all common blocks, opens some files and defines the buffer array IBUFFR with a standard size depending on the installation of SEPRAN.
SEPSTR must be called before all other SEPRAN subroutines.

Firstly the mesh that has been created by SEPMESH is read. Next the problem definition and the types of boundary conditions (not the actual values) are read and stored in array KPROB. In order to know what kind of problem definition must be given in the case of standard problems, the user must consult the manual STANDARD PROBLEMS. Finally the type of storage scheme is read, and the structure of the large matrix is computed and stored in array INTMAT.

The call of SEPSTR replaces the call of the subroutines START (4.3) (4.3), MESHRD (3.4), PROBDF (4.4) and COMMAT (4.5.2).

Heading

```
subroutine sepstr ( kmesh, kprob, intmat )
```

Parameters

integer kmesh(∗), kprob(∗), intmat(5)

kmesh Standard SEPRAN array in which information of the mesh will be stored. KMESH is a variable length array, hence position 1 must be filled by the user, with exactly the length of the declaration.
In general a length of 1000 sufficient.

kprob Standard SEPRAN array containing information concerning the problem definition. KPROB is a variable length array, hence position 1 must be filled by the user, with the declared length.
In general a length of 2000 per problem to be solved is sufficient.

intmat Standard SEPRAN array containing information concerning the structure of the system of equations to be solved.
In the case of one problem to be solved, INTMAT must be a one-dimensional array of length 5 (or larger). In the case of more problems to be solved, INTMAT should be a two-dimensional array of length 5 $\times$ n (INTMAT(5,n)), with n at least equal to the number of problems to be solved.

Input

SEPSTR requires two types of input

(i) a file named meshoutput containing the output of the mesh generator SEPMESH. meshoutput has reference number IREF10 (common block CREFS).

(ii) Input from the standard input file. This input must be given in the prescribed sequence, since it will be read sequentially. The input to be read consists of the input parts described in the Users Manual corresponding to the following keywords:

```
START   (See Users Manual Section 3.2.1)
PROBLEM (See Users Manual Section 3.2.2)
MATRIX  (See Users Manual Section 3.2.4)
```

Output

The arrays KMESH, KPROB and INTMAT have been filled.
Some information is written to a file named sepcomp.inf with reference number IREF73 (common block CREFS).

4.2.2 Subroutine SEPSTL

Description

SEPSTL is completely completely identical to SEPSTR (4.2.1), except that it allows for larger values of nbuffr to be declared by the user.

Heading

```
subroutine sepstl ( kmesh, kprob, intmat, nbuf )
```

Parameters

integer kmesh(∗), kprob(∗), intmat(5), nbuf

kmesh, kprob, intmat See subroutine SEPSTR (4.2.1).

nbuf The by the user declared length of array IBUFFR. If this length is smaller than the default length defined at the installation of SEPRAN, the default length is used. Hence if NBUF = 0, SEPSTN (4.2.3) is exactly the same as SEPSTR (4.2.1).

Input

See SEPSTR (4.2.1).

Output

See SEPSTR (4.2.1).

4.2.3 Subroutine SEPSTN

Description

SEPSTN performs exactly the same tasks as SEPSTL (4.2.2). However, with SEPSTN it is also possible to read all SEPRAN input at once and store it internally. This makes the input file more flexible and besides that makes it possible to separate reading of SEPRAN input and user input.

Heading

```
subroutine sepstn ( kmesh, kprob, intmat, iinstr )
```

Parameters

integer kmesh(∗), kprob(∗), intmat(5), iinstr(∗)

kmesh, kprob, intmat See subroutine SEPSTR (4.2.1.

iinstr Input array defined by the user. The user is himself responsible for the declaration, no checks are carried out. Contents:

Pos. 1: ILAST Last entry number that has been filled by the user. If IINSTR(1) = 0 or 1 only defaults are used.

Pos. 2: IREAD IREAD Indicates if all SEPRAN input until end of file or `end_of_sepran_input` is read (1) or that only the input as described for SEPSTR (4.2.1 is read (0)
Default value: 0

Pos. 3: NBUF If > 0, length of the buffer array, if 0, the default length is used.
Default value: 1

Input

See SEPSTR (4.2.1).
Once all input from the standard input file corresponding to the input blocks START (4.3) and PROBLEM have been read, all other input blocks are read in arbitrary sequence. Only input blocks described in Section 3.2 of the Users Manual may be read in this way.
The description is exactly the one in the Users Manual. In fact program SEPCOMP uses subroutine SEPSTN itself.

Output

See SEPSTR (4.2.1).
The input read from the input file is stored internally.

4.2.4 Subroutine SEPSTM

Description

SEPSTM performs exactly the same tasks as SEPSTN (4.2.3). SEPSTM is able to read the file `meshoutput`, but also to create a mesh itself. Whether a mesh is created or read, depends on the user input. If the user includes the input block "MESH" before the input block "PROBLEM" the mesh input is read and the mesh is created. Otherwise the mesh is read from the file `meshoutput`.
At the users option the mesh created may be written to the file meshoutput.
SEPSTM may for example be used if the user wants to run the job in one call instead of the sequence sepmesh, sepcomp, seppost. Another application is free or moving boundary problems in which extra information concerning the mesh may be necessary.

Heading

```
subroutine sepstm ( kmesh, kprob, intmat, iinstr )
```

Parameters

integer kmesh(∗), kprob(∗), intmat(5), iinstr(∗)

kmesh, kprob, intmat See subroutine SEPSTR (4.2.1).

iinstr Input array defined by the user. The user is himself responsible for the declaration, no checks are carried out. Contents:

Pos. 1: ILAST Last entry number that has been filled by the user. If IINSTR(1) = 0 or 1 only defaults are used.

Pos. 2: IREAD IREAD Indicates if all SEPRAN input until end of file or `end_of_sepran_input` is read (1) or that only the input as described for SEPSTR (4.2.1 is read (0)
Default value: 0

Pos. 3: NBUF If > 0, length of the buffer array, if 0, the default length is used.
Default value: 1

Pos. 4: WRITE_MESH Indicates if the mesh must be written to the file `meshoutput` or not.
Possible values:

0 The mesh is not written to a file.

1 The mesh is written to the file `meshoutput`.

2 The mesh is not written but the file `meshoutput` is opened.

Default value: 0

Input

See SEPSTN (4.2.3).
The input from the standard input file must contain an extra part defining the mesh. This part is exactly the input as described for program SEPMESH in the Users Manual.

The sequence of the input is the following:

```
start (optional)
 .
 .
end
mesh (mandatory)
 .
 .
 .
end
problem (mandatory)
 .
 .
 .
end

Rest of input in arbitrary sequence
```

Output

See SEPSTR (4.2.1).
The input read from the input file is stored internally.
At the users option the file meshoutput is created.

4.3 The simple starting subroutine START

Description

Starts the process, initializes default values in the common blocks, opens some files and defines the buffer array in blank common.
START must be called in each SEPRAN program, before calling other SEPRAN subroutines.
START is machine dependent.
Subroutine START does not have to be called if subroutine SEPSTR (4.2.1 (4.2) is used.

Call

```
CALL START ( ISTART, IROTAT, IOUTP, ITIME )
```

Parameters

INTEGER ISTART, IROTAT, IOUTP, ITIME

ISTART Choice parameter consisting of three parts JFILES, JSEPBC and JREFS according to |ISTART| = |JSEPBC| + |10 × JREFS + 100 × JFILES + 10000 × IINITMD|, with the sign of ISTART equal to the sign of JSEPBC. These parameters have the following meaning:

JSEPBC Choice parameter indicating if the permanent file 2 must be opened or not. Possible values:

- 0 The permanent file 2 is not used nor created.
- 1 The permanent file 2 is used; all preceding information on this file is destroyed, or the file is a new one.
- -1 The permanent file 2 is used; all preceding information on this file is kept. This possibility may only be used when file 2 has been created by a preceding SEPRAN program.

JREFS Indication whether the files with reference numbers *iref10*, *iref73* and *iref74* (*meshoutput*, *sepcomp.inf* and *sepcomp.out*) must be opened or not. (See common block CREFS). Possible values:

- 0 These files are not opened.
- 1 These files are all opened as formatted files.
- 2 These files are all opened as unformatted files, except *sepcomp.inf* (*iref73*), which is always a formatted file.

JFILES Indication whether the files 1 and 4 must be opened or not. Possible values:

- 0 The files are opened.
- 1 The files are not opened.

IINITMD Indicates if the machine-dependent subroutine INITMD, which initializes the machine-dependent common blocks, has been called before. Possible values:

- 0 INITMD has not been called.
- 1 INITMD has been called.

The sign of ISTART is negative if JSEPBC = -1, otherwise it is positive.

IROTAT Indication whether the plots must be rotated over 90 degrees or not. Possibilities:

- 0 The plots are made such that the plotting paper used is minimal. When the length of the plot in the x-direction is larger than the length in the y-direction, and the length in x-direction is smaller than the paper width, then the plot is rotated.
- 1 The plot is not rotated.

2 The plot is always rotated over 90 degrees.

IOUTP Indicates the amount of output that is produced by SEPRAN. The amount of output increases with increasing value of IOUTP.
Available values of IOUTP are:

-1 All SEPRAN output is suppressed except error messages and the output explicitly required by the user, as for example prints of the solution.

0 See -1, however, also the input is echoed.

1 See 0, in this case also the computed array length of variable arrays is printed as well as the dimension of the large matrix, the required space of the in-core buffer array and other information like that.

2 See 1, however, in this case each time a read or write from a SEPRAN direct access file is recorded.

3 See 2, however, in this case each time a memory management subroutine is called is recorded.

ITIME Indicates if the CPU time must be printed. Possible values:

1 the computation time required for each subroutine that is called, is printed,

0 this time is not printed.

Input

IROTAT, ISTART, IOUTP and ITIME must have a value.

Output

The machine dependent parameters in the common blocks CMCDPR, CMCDPI, CBUFFR, CFILE2, CFILE3, CMCDPP and CSTAND have been initialized.
NBUFFR in common block CBUFFR has got a value.
Array IBUFFR in blank common is dimensioned.
Parameters in common blocks have got their default values.
Some files are opened.

4.4 Reading the problem definition: subroutine PROBDF

Description

In this subroutine the problem definition and the types of boundary conditions are read and stored in array KPROB. PROBDF may be called more than once with different arrays KPROB1, KPROB2, ... in order to define more problems on the same mesh. Another possibility is to use PROBDF with input for more than one problem,
Subroutine PROBDF does not have to be called if subroutine SEPSTR (4.2) is used.

Heading

```
subroutine probdf ( ichoiceprob, kprob, kmesh, iinputprob )
```

Parameters

INTEGER ICHOICEPROB, KPROB(∗), KMESH(∗), IINPUTPROB(∗)

ICHOICEPROB Choice parameter. Possibilities:

- 0 All input is read from the standard input file.
  Array IINPUTPROB is not used.
- 1 All input must be stored by the user in array IINPUTPROB.
- 2 All input is read from the standard input file and stored in array IINPUTPROB.
- 3 The previously read input is re-applied to a changed mesh.
  This option is meant for free-boundary and moving boundary problems.
  In this case array IINPUTPROB is not used, but instead the information stored in array KPROB. This information contains the input of a previous call to PROBDF.
- 10 All input is read from the standard input file and stored in array IINPUTPROB. No other actions are performed, hence array KPROB will not be filled.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB is a variable length array, hence position 1 must be filled by the user, with the declared length.
In general a length of 2000 per problem to be solved is sufficient.

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

IINPUTPROB When ICHOICEPROB = 0 or 3 this array is not used and hence may have dimension 1.
When ICHOICEPROB = 1, the user must fill this variable length array, including the first position.
When ICHOICEPROB = 2, 10 the array is filled by PROBDF. The user must fill the first position.

It is very difficult to give a general estimate of the length of array IINPUTPROB. In many problems a length of 1000 suffices. If this is not enough, SEPRAN provides an error message. The structure of array IINPUTPROB is as follows:

- 1 Declared length.
- 2-4 are not used, except pos. 3, where the actual length is set by SEPRAN.
- 5 NPROB Number of problems, if zero then one is assumed.
- 6 NTYPE maximal number of types for an element group.
- 7 NUMNATBND number of standard boundary elements.

8 NUMNATBPNT number of parts where point boundary elements are created.

9 NUMNATBCRV number of parts where curve boundary elements are created.

10 NUMNATBSRF number of parts where surface boundary elements are created.

11 NUMESSPNT number of parts where point boundary conditions are given.

12 NUMESSCRV number of parts where curve boundary conditions are given.

13 NUMESSSRF number of parts where surface boundary conditions are given.

14 JLOCALTRANSFORM Indication whether local transformations are defined (JLOCALTRANSFORM = 1) or not (JLOCALTRANSFORM = 0).

15 NUMESSNODES number of groups of nodes with essential boundary conditions.

16 NUMESSELEMENTS number of groups of elements with essential boundary conditions.

17 NUMESSGROUPS number of groups of element groups with essential boundary conditions.

18 NUMCONSGROUPS number of groups of curves with boundary conditions of the type u equals unknown constant.

19 MAXNUMRENUM: |MAXNUMRENUM| defines the maximal number of degrees of freedom that is renumbered. If MAXNUMRENUM > 0, standard renumbering is used, if MAXNUMRENUM < 0, renumbering per level.

20 IVERS (Version number of array IINPUTPROB). The description previous to 1 December 1995 is defined as version number 0.
The present version number is 6.

21-40 These positions are only used if ivers$\geq$2. In that case they have the following meaning:

21 NUMESSOBSTACLES, i.e. the number of obstacles in which essential boundary conditions are given, provided there are nodes at the obstacle.

22 NUMNATOBSTACLES, i.e. the number of obstacles in which natural boundary conditions are given, provided there are nodes at the obstacle. In that case boundary elements are defined at the obstacle curves.

23 NUMLOCOBSTACLES, i.e. the number of obstacles in which local transformations must be defined.

24 NESCONTACT, i.e. the number of contacts, where all points that make contact have essential boundary conditions

25 NUMNATCONTACTELEMENTS, i.e. the number of contacts in which natural boundary conditions are given in the form of elements, provided there are nodes in the contact surface that make contact.

26 NUMLOCCONTACTS, i.e. the number of contacts in which local transformations must be defined.

27 NESNOCONTACT, i.e. the number of contact surfaces, where all points that make no contact have essential boundary conditions

28 NUMNATCONTACTPOINTS, i.e. the number of contact points in which natural boundary conditions are given in the form of points, provided there are nodes in the contact surface that make contact.

29 NUMNATNOCONTACTELEMENTS, i.e. the number of elements in the contact surfaces that make no contact in which natural boundary conditions are given in the form of elements, provided there are nodes in the contact surface that make no contact.

30 NUMNATNOCONTACTPOINTS, i.e. the number of points that make no contact in a contact surface in which natural boundary conditions are given in the form of points, provided there are nodes in the contact surface that make no contact.

31 NUMNATCURPOINTS, i.e. number of parts where point boundary elements are defined on curves

32 NUMNATSURPOINTS, i.e. number of parts where point boundary elements are defined on surfaces

33 NESNODAL, i.e. number of parts where essential boundary conditions defined on nodal points are given.

34 NGLGRP, i.e. the number of global element groups.

35 NUMGLCRV, i.e. the number of curve groups where global elements are defined.

36 NUMGLSRF, i.e. the number of surface groups where global elements are defined.

37 NUMGLVLM, i.e. the number of volume groups where global elements are defined.

38 NUMGLALL, i.e. the number of "all" groups where global elements are defined. This means the number of times that "all" is used in the input.

39 NUMNATCROSSOBST, i.e. number of obstacles where boundary elements are defined by the cross section of the obstacle with a fixed grid.

40 PRINT_LEVEL, i.e. level of print information.

41-60 These positions are only used if ivers≥5. In that case they have the following meaning:

41 NFICTGRP, i.e. the number of fictitious element groups.

42 NUMFICTCRV, i.e. the number of curve groups where fictitious elements are defined.

43 NUMFICTSRF, i.e. the number of surface groups where fictitious elements are defined.

44 NUMESSVLM, i.e. number of volumes with essential boundary conditions.

45 NUMFICTOBST, i.e. number of obstacles where fictitious elements are defined.

46 NUMESSINNOBST, i.e. number of obstacles with essential boundary conditions in all nodes of the elements of the mesh that are completely within the obstacle.
See Users Manual Section 2.2: IN_INNER_OBSTACLE.

47 NUMESSBOUNOBST, i.e. number of obstacles with essential boundary conditions in all nodes of the elements of the mesh that are partly within the obstacle.
See Users Manual Section 2.2: IN_BOUN_OBSTACLE.

48 NUMESSONBOUNOBST, i.e. number of obstacles with essential boundary conditions in all nodes of the mesh that are on the boundary of the obstacle.
See Users Manual Section 2.2: ON_BOUN_OBSTACLE.

49 ISKIPELEM. If this parameter is larger than zero this means that information concerning the skipping of elements is stored.

50 NUMESSALLOBST, i.e. number of obstacles with essential boundary conditions in all nodes of the mesh that are within the obstacle

51 NUMESSINONBOUNOBST, i.e. number of obstacles with essential boundary conditions in all nodes of the mesh that are on the boundary of the obstacle and also only in elements that are inside the obstacle.
See Users Manual Section 2.2: INON_BOUN_OBSTACLE.

52 NESZEROLEV, i.e. number of level set parts where we have essential boundary conditions for zero level sets.

53 NUMLEVELSET, i.e. number of level sets

54 NESCAV

55 NATLEVSET, i.e. number of natural boundary conditions at the zero levelset boundary.

56 NESINTERSECT, i.e. number of intersections where essential boundary conditions are given.

57-60 Not yet defined.

Next positions: information of the type numbers .

For each element group:

NTYPE positions with type numbers.

When all type numbers are between 1 and 100, followed by:

(i) INUNKP number of degrees of freedom in the nodal points of the element.
If the number of degrees of freedom in each nodal point of the element is not constant, INUNKP must be equal to zero.

(ii) INDTVC number of vectors of special structure.

(iii) ISEAR: number of specific degrees of freedom. If ISEAR = 0 the i^{th} degree of freedom in each nodal point corresponds to the same physical variable in each node. Otherwise ISEAR (>0) gives the number of different physical variables.

(iv) If INUNKP = 0, INPELM (is number of nodal points in the element) extra positions are needed, containing the number of degrees of freedom in the nodal points.

(v) If INDTVC > 0, INDTVC groups of INPELM extra positions are needed, containing the number of degrees of freedom in the nodal points of the i^{th} vector of special structure.

(vi) If ISEAR > 0, ISEAR groups of INPELM extra positions are needed, containing the position of the physical variable in each local node of the element. (compare with the Users Manual Section 3.2.2)

When the type number is equal to -1 and the version number is at least 1 followed by:

(i) IDEGFD, i.e. degree of freedom that must be coupled. If 0 all degrees of freedom are coupled (pure periodical boundary conditions).

In the special case that standard elements with type numbers > 99 will be used and still the user wants to define some extra arrays of special structure, the type number must consist of two parts ITYPE and IEXTRA according to:
Type number = ITYPE + 10000 × IEXTRA
IEXTRA = 0 is the standard situation,
IEXTRA = 1 means that extra information is required. In that case the type numbers must be followed by the following information:

(i) INDTVC_extra, i.e. the extra number of vectors of special structure to be defined, followed by INDTVC_extra groups of 4 positions with the following information with respect to each extra vector of special structure:

1 Number of unknowns in the vertices of the elements.

2 Number of unknowns in the mid side points

3 Number of unknowns in the centroid

4 Number of unknowns in the centroids of faces (3D)

Next positions: information of the boundary elements .

For each boundary element group:

NTYPE positions with type numbers.
When all type numbers are smaller than 100, followed by:

(i) INPELM: number of nodal points in element.

(ii) INUNKP, i.e. number of unknowns per point.
Possible values:

-1 The same number of degrees of freedom as in the corresponding internal elements is used.

0 INPELM (is number of nodal points in the element) extra positions are needed, containing the number of degrees of freedom in the nodal points.

>0 Number of unknowns per node.

(iii) INDTVC, i.e. number of vectors of special structure.
If 0 the same as for the internal elements is assumed.

(iv) ISEAR, i.e. number of specific degrees of freedom per point.
If 0 the same as for the internal elements is assumed.

(v) If INUNKP = 0, INPELM positions with the number of degrees of freedom per node.

(vi) If INDTVC > 0, information about the vectors of special structure in exactly the same way as for the internal elements.

(vii) If ISEAR > 0, information about the physical degrees of freedom in exactly the same way as for the internal elements.

Remark: The positions (ii) to (vii) are only used if IVERS $\geq$ 3, otherwise only position (i) is used.

This information must be followed by:

NUMNATBPNT groups with information of the point boundary elements:

(i) Boundary element group number.

(ii) Number of user points.

(iii) User point numbers.

NUMNATBCRV groups with information of the curve boundary elements:

(i) Boundary element group number.

(ii) Element shape number, possibilities:

1 linear elements

2 quadratic elements

(iii) IFIRST: first curve number for boundary elements of this group.

(iv) ILAST: last curve number for boundary elements of this group.

NUMNATBSRF groups with information of the surface boundary elements:

(i) Boundary element group number.

(ii) IFIRST: first surface number for boundary elements of this group.

(iii) ILAST: last surface number for boundary elements of this group.

NUMNATOBSTACLES groups with information of the obstacle boundary elements:

(i) Boundary element group number.

(ii) Element shape number of boundary element according to Table 2.2 in the Users Manual.

(iii) Sequence number of obstacle at which boundary elements must be defined.

(iv) itypeskip, defines if obstacle curves must be skipped for the definition of the cross sections (1) or not (0)

(v) ifirst, first number of curve to be skipped if itypeskip = 1

(vi) ilast, last number of curve to be skipped if itypeskip = 1

NUMNATCONTACTELEMENTS groups with information of the boundary elements that make complete contact and in which natural boundary conditions are given:

(i) Boundary element group number.

(ii) Element shape number of boundary element according to Table 2.2 in the Users Manual.

(iii) Sequence number of contact at which boundary elements must be defined.

NUMNATNOCONTACTELEMENTS groups with information of the boundary elements that make no contact but are in a surface marked as contact surface and in which natural boundary conditions are given:

(i) Boundary element group number.

(ii) Element shape number of boundary element according to Table 2.2 in the Users Manual.

(iii) Sequence number of contact at which boundary elements must be defined.

NUMNATCONTACTPOINTS groups with information of the boundary point elements that make contact and in which natural boundary conditions are given. Hence in this case it concerns all points in the contact surface that make contact. The elements are automatically point elements.

(i) Boundary element group number.

(ii) 0 to indicate that it concerns a point boundary condition.

(iii) Sequence number of contact at which boundary elements must be defined.

NUMNATNOCONTACTPOINTS groups with information of the boundary point elements that make no contact but are in a surface marked as contact surface and in which natural boundary conditions are given. Hence in this case it concerns all points in the contact surface that make contact. The elements are automatically point elements.

(i) Boundary element group number.

(ii) 0 to indicate that it concerns a point boundary condition.

(iii) Sequence number of contact at which boundary elements must be defined.

NUMNATCURPOINTS groups with information of the point boundary elements defined on curves:

(i) Boundary element group number.

(ii) IFIRST: first curve number for boundary elements of this group.

(iii) ILAST: last curve number for boundary elements of this group.

NUMNATSURPOINTS groups with information of the point boundary elements defined on surfaces:

(i) Boundary element group number.

(ii) IFIRST: first surface number for boundary elements of this group.

(iii) ILAST: last surface number for boundary elements of this group.

NUMNATCROSSOBST groups with information of the special boundary elements defined as cross section of a fixed mesh with an obstacle:

(i) Boundary element group number.

(ii) IOBST: obstacle sequence number for boundary elements of this group.

(iii) IEXCLUDE: defines which elements are taken into account and which are skipped in order to prevent singular matrices.
Possible values

(a) Only one cross-section boundary element can be coupled to each node outside the obstacle.

(b) Only one cross-section boundary element can be coupled to each node of the mesh.

Next positions: information of the global element groups , provided NGLGRP > 0.

NGLGRP groups of 2 positions each containing

(i) ITYPE, i.e. type number for the group

(ii) NUMDEGFD, i.e. number of global unknowns for this group

Next NUMGLCRV groups with information about the definition of global elements on curves:

(i) Global element group number

(ii) ISHAPE, i.e. shape number of the elements

(iii) ICRV1, i.e. first curve number

(iv) ICRV2, i.e. last curve number

Next NUMGLSRF groups with information about the definition of global elements on surfaces:

(i) Global element group number

(ii) ISRF1, i.e. first surface number

(iii) ISRF2, i.e. last surface number

Next NUMGLVLM groups with information about the definition of global elements on volumes:

(i) Global element group number

(ii) IVLM1, i.e. first volume number

(iii) IVLM2, i.e. last volume number

Next NUMGLALL groups with information about the definition of global elements on "all" parts:

(i) Global element group number

Finally information about the global renumbering in the sequence:

(i) NGLOBRENUM = Number of global unknowns to be renumbered

(ii),... NGLOBRENUM sequence numbers of global unknowns to be renumbered

Next positions: information of the fictitious element groups , provided NFICTGRP > 0.

NFICTGRP groups of 2 positions each containing

(i) ITYPE, i.e. type number for the group

(ii) NUMDEGFD, i.e. number of global unknowns for this group

Next NUMFICTCRV groups with information about the definition of fictitious elements on curves:

(i) Fictitious element group number

(ii) ISHAPE, i.e. shape number of the approximation for the multipliers

(iii) ICRV1, i.e. first curve number

(iv) ICRV2, i.e. last curve number

(v) ISTRUCGROUP, i.e. element group number for the structural elements.

(vi) IFLUIDGROUP1, i.e. first element group number for the fluid elements.

(vii) IFLUIDGROUP2, i.e. last element group number for the fluid elements.

Next NUMFICTSRF groups with information about the definition of fictitious elements on surfaces:

(i) Fictitious element group number

(ii) ISHAPE, i.e. shape number of the approximation for the multipliers
(iii) ISRF1, i.e. first surface number
(iv) ISRF2, i.e. last surface number
(v) ISTRUCGROUP, i.e. element group number for the structural elements.
(vi) IFLUIDGROUP1, i.e. first element group number for the fluid elements.
(vii) IFLUIDGROUP2, i.e. last element group number for the fluid elements.

Next positions: information of the essential boundary conditions .

NUMESSPNT groups with information of the points where essential boundary conditions are given:
(i) Bit pattern IBIT, see IBIT.
(ii) NUMBPN: Number of user points.
(iii) User point numbers (NUMBPN numbers).

NUMESSCRV groups with information of the curves where essential boundary conditions are given:
(i) Bit pattern IBIT, see IBIT.
(ii) JSTEP: Indication of which points on the curves have prescribed degrees of freedom. Possibilities:
JSTEP=0 all nodal points on the curves are prescribed as indicated by IBIT.
JSTEP>0 only the points 1, 1+JSTEP, 1+2×JSTEP, . . . on these curves are prescribed as indicated by IBIT.
JSTEP<0 all nodal points except the points 1, 1-JSTEP, 1-2×JSTEP, . . . on these curves are prescribed by IBIT.
(iii) IFIRST: first curve number for this group.
If IFIRST = -1, all outer curves are used.
(iv) ILAST: last curve number for this group.

NUMESSSRF groups with information of the surfaces where essential boundary conditions are given:
(i) Bit pattern IBIT, see IBIT.
(ii) IFIRST: first surface number for this group.
If IFIRST = -1, all outer surfaces are used.
(iii) ILAST: last surface number for this group.
(iv) INPELM positions corresponding to the nodal points of the elements. When position j equals zero, then the nodal point is skipped, when position j equals one, then the nodal point has the prescribed boundary conditions as indicated by IBIT.

If the version number IVERS $\geq$ 6 then these positions are followed by at least 2 extra positions:
(v) ISKIP_BOUNDARY defines if curves must be skipped and which curves. Possible values:

0 means that the complete surface including boundary points is taken into account.
1 means that only the inner points of the set of surfaces is used.
2 means that only the inner points of the set of surfaces is used, plus all outer curves that are given explicitly. These curves are defined in part (vii).
3 means that only the inner points of the set of surfaces is used, plus all outer curves except those that are given explicitly. These curves are defined in part (vii).

(vi) NCURVES_EXTRA Number of curves in part (vii)

(vii) NCURVES_EXTRA curves

NUMESSVLM groups with information of the volumes where essential boundary conditions are given:

(i) Bit pattern IBIT, see IBIT.

(ii) IFIRST: first volume number for this group.

(iii) ILAST: last volume number for this group.

(iv) INPELM positions corresponding to the nodal points of the elements. When position j equals zero, then the nodal point is skipped, when position j equals one, then the nodal point has the prescribed boundary conditions as indicated by IBIT.

(v) ISKIP_BOUNDARY defines if surfaces must be skipped and which surfaces.
Possible values:

0 means that the complete volume including boundary points is taken into account.
1 means that only the inner points of the set of volumes is used.
2 means that only the inner points of the set of volumes is used, plus all outer surfaces that are given explicitly. These surfaces are defined in part (vii).
3 means that only the inner points of the set of volumes is used, plus all outer surfaces except those that are given explicitly. These surfaces are defined in part (vii).

(vi) NSURFS_EXTRA Number of surfaces in part (vii)

(vii) NSURFS_EXTRA surfaces

NUMESSNODES groups with information of the nodes where essential boundary conditions are given:

(i) Bit pattern IBIT, see IBIT.

(ii) IFIRST: first nodal point number for this group.

(iii) ILAST: last nodal point number for this group.

NUMESSELEMENTS groups with information of the elements where essential boundary conditions are given in a relative nodal point:

(i) Bit pattern IBIT, see IBIT.

(ii) IFIRST: first element number for this group.

(iii) ILAST: last element number for this group.

(iv) RN1: relative nodal point number in these elements. $1 \leq \text{RN1} \leq \text{INPELM}$.

NUMESSGROUPS groups with information of the element groups where essential boundary conditions are given in the nodal points:

(i) Bit pattern IBIT, see IBIT.

(ii) IELGRP: element group number.

(iii) INPELM positions corresponding to the nodal points of the elements. When position j equals zero, then the nodal point is skipped, when position j equals one, then the nodal point has the prescribed boundary conditions as indicated by IBIT.

NUMESSOBSTACLES groups with information of the obstacles at which essential boundary conditions are given in the nodal points which are lying on the obstacle:

(i) Bit pattern IBIT, see IBIT.

(ii) JSTEP (see NUMESSCRV)

(iii) Obstacle sequence number.

NUMESSALLOBST groups with information of the obstacles where essential boundary conditions are given in all nodal points of the mesh which are lying in the obstacle:

(i) Bit pattern IBIT, see IBIT.

(ii) Obstacle sequence number.

NUMESSINNOBST groups with information of the obstacles where essential boundary conditions are given in all nodal points of the elements of the mesh which are lying completely in the obstacle:

(i) Bit pattern IBIT, see IBIT.

(ii) Obstacle sequence number.

NUMESSBOUNOBST groups with information of the obstacles where essential boundary conditions are given in all nodal points of the elements of the mesh which are lying partly in the obstacle:

(i) Bit pattern IBIT, see IBIT.

(ii) Obstacle sequence number.

NUMESSONBOUNOBST groups with information of the obstacles where essential boundary conditions are given in all nodal points of the mesh which are lying on the boundary of the obstacle:

(i) Bit pattern IBIT, see IBIT.

(ii) Obstacle sequence number.

NUMESSINONBOUNOBST groups with information of the obstacles where essential boundary conditions are given in all nodal points of the mesh which are lying on the boundary of the obstacle and which are inside the obstacle:

(i) Bit pattern IBIT, see IBIT.

(ii) Obstacle sequence number.

NESCONTACT groups with information of the contacts at which essential boundary conditions are given in the nodal points that make contact:

(i) Bit pattern IBIT, see IBIT.

(ii) JSTEP (see NUMESSCRV)

(iii) Contact sequence number.

NESNOCONTACT groups with information of the contacts at which essential boundary conditions are given in the nodal points that make no contact:

(i) Bit pattern IBIT, see IBIT.

(ii) JSTEP (see NUMESSCRV)

(iii) Contact sequence number.

NESNODAL groups with information of nodal points on which essential boundary are given:

(i) Bit pattern IBIT, see IBIT.

(ii) NUMNODAL: if positive then: number of nodal points for this group. If this number is negative the absolute value denotes the number of nodal points, and it is assumed that the nodal point numbers are stored in a user provided file. This option corresponds to the option `FILE_NODAL_POINTS = 'name_of_file'` in the input file as described in the Users Manual Section 3.2.3.

If NUMNODAL > 0

(iii) to (ii+NUMNODAL) Nodal point numbers.

else NUMNODAL < 0

(iii) IPOS, i.e. position of name of file in the array TEXTS in common block CSEPCM (21.19).

Table 4.1.2 Bit patterns.

bit pattern		meaning
dec.	binary	
1	1	first unknown prescribed
2	10	second unknown prescribed
3	11	first and second unknown prescribed
4	100	third unknown prescribed
5	101	first and third unknown prescribed
6	110	second and third unknown prescribed
7	111	first, second and third unknown prescribed

IBIT Bit pattern, contains information which of the degrees of freedom are prescribed in the nodal point.
When IBIT = 0, all degrees of freedom in the nodal point are prescribed.
Otherwise the following rules must be applied:
When the j^{th} unknown in a nodal point is prescribed, the corresponding bit is equal to one, if it is not prescribed, it is equal to zero.
Bits are numbered from **right** to **left**.

Example

Let nodal point i have 8 unknowns, with the first, the third, the fourth, the sixth, the seventh and the eight prescribed.
bit pattern:

1	1	1	0	1	1	0	1
8	7	6	5	4	3	2	1

Converting this bit pattern to decimal gives:

$$\text{dec} = \sum_i bit(i) \times 2^{i-1}$$

In this case $2^0 + 2^2 + 2^3 + 2^5 + 2^6 + 2^7 = 237$
Hence the bit pattern is equal to 237.

The most common bit patterns are given in table 4.1.2

Next positions: information of the local transformations , when JLOCALTRANSFORM = 1:

Pos. 1 NUMLOCALCRV number of parts where local transformations on curves are given.

Pos. 2 NUMLOCALSRF number of parts where local transformations on surfaces are given.

NUMLOCALCRV groups with information of the curves where local transformations
are defined.

(i) JDEGFD1.

(ii) JDEGFD2.

(iii) JSTEP: indication of which points on the curves must be transformed. Possibilities:

JSTEP=0 All nodal points on the curves are transformed as indicated by JDEGFD1 and JDEGFD2.

JSTEP>0 Only the points 1, 1 + JSTEP, 1 + 2 × JSTEP, . . . on these curves are transformed.

JSTEP<0 All points except 1, 1 - JSTEP, 1 - 2 × JSTEP, . . . on these curves are transformed.

(iv) IFIRST: first curve number for this group (>0 or <0).

(v) ILAST: last curve number for this group (>0 or <0).

(vi) ITRANS, defines how the transformation matrix must be computed. Possible values:

1 Standard situation, the transformation matrix **R** is computed using the normal vector as defined in the Users Manual Section 3.2.2 corresponding to STANDARD.

2 The option SYMMETRIC is used.

3 The option NON_SYMMETRIC is used.

4 The option MIN_UN_UT is used.

If ITRANS=2, 4 extra positions are required defining the entries of the transformation matrix. These extra positions may be positive or negative.

A negative value defines -IFUNC, which implies that function subroutine FUNCTR (User Manual 3.3.7) is called with parameter ICHOIS_TRANS equal to IFUNC.

A positive value (IPOS) means that the entry is a constant stored in array RINPUT position IPOS.

Array RINPUT is not directly available to the user, because there is no such parameter in PROBDF. However, the array is stored in the buffer array IBUFFR and may be indirectly addressed through array KPROB. See Section 24.3.

NUMLOCALSRF groups with information of the surfaces where local transformations
are defined.

(i) JDEGFD1.

(ii) JDEGFD2.

(iii) JDEGFD3.

(iv) IFIRST: first surface number for this group (>0 or <0).

(v) ILAST: last surface number for this group (>0 or <0).

(vi) ITRANS, defines how the transformation matrix must be computed. Possible values:

1 Standard situation, the transformation matrix **R** is computed using the normal vector and tangential vectors as defined in the Users Manual Section 3.2.2 corresponding to STANDARD.

2 The option SYMMETRIC is used.

3 The option NON_SYMMETRIC is used.

If ITRANS=2, 4 extra positions are required defining the entries of the transformation matrix. They are defined in exactly the same way as for curves, only the size of the matrices is 3 × 3 instead of 2 × 2.

(vii) ISKIP_NORMAL. This parameters consists of two parts: ISKIP_BOUNDARY and NORMAL_DIRECTION, according to
ISKIP_NORMAL = ISKIP_BOUNDARY +10 × NORMAL_DIRECTION.
These two parameters have the following meaning:

ISKIP_BOUNDARY defines if curves must be skipped and which curves.
Possible values:

0 means that the complete surface including boundary points is taken into account.
1 means that only the inner points of the set of surfaces is used.
2 means that only the inner points of the set of surfaces is used, plus all outer curves except those that are given explicitly. These curves are defined in part (xi). 3 means that only the inner points of the set of surfaces is used, plus all outer curves that are given explicitly. These curves are defined in part (xi).

NORMAL_DIRECTION = 0, means that the outward directed normal is used to define the first direction.
1 means that the inwards directed normal is used.

(viii) TANG, defines how the tangential vector is computed.
TANG = 0, implies that SEPRAN decides itself how to compute these vectors, which means that the user does not know how they are defined.
TANG = 1, means that the first tangential direction is defined by the straight line through the points P1 and P2.
TANG = 2, 3, 4, means that the first tangential direction is defined by the unit vector $\mathbf{e}_t$, with t equal to TANG-1.
TANG = 10, means that the second tangential direction is along a set of curves. This option is meant to define a solution parallel to this set of curves, by describing the first two degrees of freedom. Since these are both orthogonal to the third one, this defines the $flow$ parallel to the curves.

(ix) INPELM positions corresponding to the nodal points of the elements.
When position j equals zero, then the nodal point is skipped, when position j equals one, then a local transform must be carried out in the nodal point.

(x) If TANG = 1, 2 extra positions containing the user point sequence numbers of the points P1 and P2, with respect to the line defining the tangential vector.
If TANG = 10, we need $numcurves$+1 extra positions, where $numcurves$ is the number of curves along which we want to define the second tangential direction. The first of these extra positions contains $numcurves$ and the other ones the corresponding curve numbers.

(xi) is only used in case of ISKIP_BOUNDARY = 2 or 3. In that case the length of this part is ncurves_local+1, where ncurves_local is the number of outer curves that our given explicitly. The first position of this part contains ncurves_local, and the following positions the corresponding curve numbers.

NUMLOCOBSTACLES groups with information of the obstacle curves where local transformations
are defined.

(i) JDEGFD1.

(ii) JDEGFD2.

(iii) -

(iv) JSTEP: see curves.

(v) Obstacle sequence number

(vi) ITRANS, defines how the transformation matrix must be computed. See curves.

NUMLOCCONTACTS groups with information of the contact curves where local transformations
are defined.

(i) JDEGFD1.
(ii) JDEGFD2.
(iii) -
(iv) JSTEP: see curves.
(v) Contact sequence number
(vi) ITRANS, defines how the transformation matrix must be computed. See curves.

information where the boundary condition u equals unknown constant is given
when NUMCONSGROUPS > 0.

NUMCONSCRV number of parts of curves that contain this type of boundary condition.

NUMCONSSRF number of parts of surfaces that contain this type of boundary

NUMCONSCRV groups with information of the curves where the boundary condition
u equals unknown constant is given.
(i) DEGFDl.
(ii) JSTEP: indication of which points contain the specific boundary condition. (JSTEP is equal to i in the input file). Possibilities:
JSTEP=0 All nodal points on the curves contain the degree of freedom DEGFDl.
JSTEP>0 Only the points 1, 1 + JSTEP, 1 + 2 × JSTEP, . . . on these curves contain the degree of freedom DEGFDl.
JSTEP<0 All points except 1, 1 - JSTEP, 1 - 2 × JSTEP, . . . on these curves contain the degree of freedom DEGFDl.
(iv) IFIRST: first curve number for this group (>0 or <0).
(v) ILAST: last curve number for this group (>0 or <0).

NUMCONSSRF groups with information of the surfaces where the boundary condition
u equals unknown constant is given.
(i) DEGFDl.
(ii) IFIRST: first surface number for this group.
(iii) ILAST: last surface number for this group.
(iv) INPELM: number of points in the elements of the surfaces.
(v) INPELM positions corresponding to the nodal points of the elements. When position j equals zero, then the nodal point is skipped, when position j equals one, then the nodal point has the prescribed boundary conditions as indicated by IBIT.

|MAXNUMRENUM| positions with the sequence numbers of the degrees of freedom
with respect to renumbering as given by the command RENUMBER.
Equal sequence numbers mean that these degrees of freedom are numbered at the same level.
For example, the statement

RENUMBER 3, 4, (5, 6), 2, 1 gives the following 6 positions:

5, 4, 1, 2, 3, 3

Remark: the sequence numbers must be numbered consecutively from sequence number 1.

Next positions information about elements to be skipped

If ISKIPELEM > 0, this part is used.
It contains all information referring to the skipping of elements.
The first position contains the number of parts that are used to define where elements are to be skipped (NSKIP).
The next NSKIP groups of 5 positions each contain information depending on how the skipping is defined.
Position 1 contains ITYPE_SKIP, which defines the type of skipping.
Possible values

1. All elements that are completely within obstacle *iobst* are skipped.
   Position 2 contains *iobst* and positions 3 to 5 are not used.
2. All elements that have only points that are either in obstacle *iobst* or on the curves c*i* to C*j* are skipped.
   Position 2 contains *iobst*, position 3 curve number i, position 4 curve number j, and position 5 is not used.

If more than one problem must be stored in KPROB, then for each next problem the positions 6, 7, ... etc. must be repeated after the first problem.

Input

ICHOICEPROB must have a value.
Array KMESH must have been filled.
Depending on the value of ICHOICEPROB, input is read from the standard input file (ICHOICEPROB = 0,2) or from array IINPUT (ICHOICEPROB = 1).

Output

Array KPROB has been filled.
When ICHOICEPROB = 2 or 10, array IINPUTPROB has been filled.

Remarks

An alternative of subroutine PROBDF is subroutine PROBDFBF. This subroutine is identical to PROBDF, except for two extra parameters IBUFFR and BUFFER, that are used as first and second parameter. In this way the local use of blank common is avoided.

Heading:

```
subroutine probdfbf ( ibuffr, buffer, ichoiceprob, kprob, kmesh,
                      iinputprob )
```

4.5 Definition of the matrix structure: subroutine MATSTRUC

In this section we treat the SEPRAN subroutines that may be used to define the structure of the matrix. At this moment there are two subroutines available:

MATSTRUC (4.5.1) this is the standard subroutine for creating the structure of the matrix. It allows for all possibilities described in the Users Manual Section 3.2.4.

COMMAT (4.5.2 (4.5.2) is the old subroutine that creates the structure of the matrix. Not all possibilities available in Section 3.2.4 of the Users Manual are present in COMMAT (4.5.2.

4.5.1 Subroutine MATSTRUC

Description

Integer information concerning the large matrix is computed.
The solution method is chosen.
Subroutine MATSTRUC does not have to be called if one the starting subroutines of Section 4.2 (SEPSTR (4.2.1, SEPSTL (4.2.2, SEPSTN (4.2.3, SEPSTM (4.2.4) is used.

Call

```
CALL MATSTRUC ( IINCOMMT, KMESH, KPROB, INTMAT )
```

Parameters

INTEGER IINCOMMT(∗), KMESH(∗), KPROB(∗), INTMAT(5)

IINCOMMT Integer input array that must have been filled by the user with information concerning the structure of the large matrix.
IINCOMMT must be filled as follows:

1 Highest entry number of IINCOMMT that has been filled by the user. The positions 1 to IINCOMMT(1) must have been given a value by the user.
For all other positions default values are used.

2 JMETHOD parameter indicating the structure of the large matrix. This parameter is the same as the parameter i in METHOD $= i$ as described in the Users Manual Section (3.2.4), input block MATRIX. It is a combination of all possible options defined by the various keywords in that section.
Possible values:

1 The matrix is a symmetric profile matrix.

2 The matrix is a non-symmetric profile matrix.

3 The matrix is a complex symmetric profile matrix.

4 The matrix is a complex non-symmetric profile matrix.

5 The matrix is stored as a compact matrix, i.e. all essential zero elements are skipped. This storage can only be used for matrix-vector multiplication and iterative solution techniques. (symmetric matrix and symmetric storage).

6 See 5, however, for non-symmetric matrices. The integer description is stored symmetrically, i.e., only the indices of the lower triangle are stored.

7 See 5, however, for symmetric complex matrices.

8 See 6, however, for general complex matrices.

9 The matrix is stored as a compact matrix, i.e. all essential zero elements are skipped. This storage can only be used for matrix-vector multiplication and iterative solution techniques. The storage is non-symmetrical and therefore requires more space than the storage schemes 5 and 6.

10 See 9, however, for complex matrices.

11 Combined row/column compressed storage scheme, designed for iterative solvers on vector processors. Moreover, an internal permutation of rows and columns is used. It stores the whole off-diagonal part of the matrix.

It has no effect on scalar processors.

12 See 11, however, for complex matrices.

13 Special storage for simple type solvers (only for Navier-Stokes). The velocity matrix is symmetric and the divergence matrix and gradient matrix are each others transpose.

14 See 13, however, the velocity matrix is non-symmetric.

15 See 13, however, the divergence matrix and the gradient matrix are different, although their structure is the same.

16 See 14, however, the divergence matrix and the gradient matrix are different, although their structure is the same.

19 The matrix is a hermitian complex profile matrix.

20 The matrix is full, symmetric and real.

21 The matrix is full, non-symmetric and real.

22 The matrix is full, symmetric and complex.

23 The matrix is full, non-symmetric and complex.

24-27 See 5 to 8, however, the shifted Laplace storage is used.

31-34,49 See JMETHOD = 1 to 4, 19. In this case the program stops if the matrix does not fit into memory. In the usual case a warning is given and the matrix is split into parts that are put onto disk. Splitting of the matrix has a negative influence on the performance.

51 row compact storage for amg

52 row compact storage for mumps (unsymmetric)

53 row compact storage for mumps (symmetric)

54 row compact storage for mumps (symmetric, pos def)

55 row compact storage for petsc

56 row compact storage for petsc, no sepran mat

For JMETHOD = 1,2,3, 4 or 19 a profile (also called skyline) storage is used. This storage reduces computing time and memory compared to a band method.
Default value: 2

3 IPRINT defines the print level. The following values if IPRINT are allowed:

0 no extra information is printed

1 The contents of array INTMAT corresponding to the internal unknowns is printed

2 The complete contents of array INTMAT is printed

10-12 See 0-2, however also the structure of the matrix is printed formally. These values are only available for JMETHOD=1-4.

Default value: 0

4 INVMAT_TOT consists of two parts: INVMAT and IBCMAT according to INVMAT_TOT = INVMAT+2×IBCMAT with The value of INVMAT indicates if not only the standard matrix must be assembled, but also a matrix that is assembled from element matrices that are created by computing the LU decomposition of the element matrix. This last matrix may be used as preconditioner for for example Conjugate Gradient solvers. Possible values of INVMAT are:

0 Standard situation, the extra matrix is not computed.

1 The extra matrix is built, using the LU-decompositions of the element matrices.

INVMAT = 1 may only be used in combination with JMETHOD = 5 - 8.
The value of IBCMAT indicates if also the matrix of reaction forces must be built as described in the User Manual Section 3.2.4. Possible values of IBCMAT are:

0 Standard situation, the extra matrix is not computed.

1 The matrix of reaction forces is built.

Default value: 0

5 IPROB The problem number for which the structure of the large matrix must be computed.
($1 \leq$ IPROB $\leq$ NPROB), where NPROB denotes the number of problems defined in the PROBLEM input.
Default value: 1

6 FILLIN_EXTRA defines if the structure of the matrix must be created such that extra fill-in is possible Possible values:

0 No extra fill-in

1 Special case for discontinuous Navier-Stokes equations. The connections of the centroids of the elements with the centroids of neighboring elements are also filled in the structure for all physical variables iphys1 to iphys2.

fillin_extra > 0 makes only sense for a compact storage in combination with a preconditioned solver.
Default value: 0

7 iphys1, see IINCOMMT(6)
Default value: $ndim + 1$, with $ndim$ the dimension of the space.

8 iphys2, see IINCOMMT(6)
Default value: If linear elements are present $ndim + 1$ otherwise $2 \times ndim + 1$

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4.

INTMAT Standard SEPRAN array containing information concerning the structure of the system of equations to be solved.

Input

Array IINCOMMT must have been filled by the user.
The arrays KMESH and KPROB must be filled.

Output

Array INTMAT has been filled.

Remark

For each new problem to be solved, a separate call of MATSTRUC is necessary. Each problem requires a new array INTMAT. The easiest way to introduce more more arrays INTMAT is to declare INTMAT as a two-dimensional array of size 5 $\times$ NPROB, with NPROB the number of problems. The call of MATSTRUC becomes:

```
CALL MATSTRUC ( IINCOMMT, KMESH, KPROB, INTMAT(1,IPROB) )
```

where INTMAT is declared as: INTMAT(5, NPROB).

4.5.2 Subroutine COMMAT

Description

Integer information concerning the large matrix is computed.
The solution method is chosen.
Subroutine COMMAT does not have to be called if subroutine SEPSTR (4.2.1 (4.2) is used.

Call

```
CALL COMMAT ( ICHCOM, KMESH, KPROB, INTMAT )
```

Parameters

INTEGER ICHCOM, KMESH(∗), KPROB(∗), INTMAT(5)

ICHCOM Choice parameter for subroutine COMMAT consisting of four parts JMETHOD, IPROB, IBCMAT and INVMAT according to
| ICHCOM | = | JMETHOD + 100 × (INVMAT + 2 × IBCMAT) + 1000 × (IPROB-1) |
with

JMETHOD Parameter to indicate which solution method is chosen. Possibilities:

> 0 See subroutine MATSTRUC

< 0 When JMETHOD < 0 then the subroutine fills array INTMAT in the same way as for -JMETHOD. However, also the structure of the large matrix is printed formally. This possibility is only available for | JMETHOD | ≤ 4. It is recommended to use it only for small problems.

INVMAT The absolute value of this parameter indicates if not only the standard matrix must be assembled, but also a matrix that is assembled from element matrices that are created by computing the LU decomposition of the element matrix. This last matrix may be used as preconditioner for for example Conjugate Gradient solvers. Possible values of | INVMAT | are:

0 Standard situation, the extra matrix is not computed.

1 The extra matrix is built, using the LU-decompositions of the element matrices. The sign of INVMAT must be equal to the sign of JMETHOD.

IBCMAT See subroutine MATSTRUC.

IPROB See subroutine MATSTRUC.

KMESH, KPROB, INTMAT See subroutine MATSTRUC.

Input

JMETHOD must have a value.
The arrays KMESH and KPROB must be filled.

Output

Array INTMAT has been filled.

4.6 Changing the type numbers from one set to another: subroutine CHTYPE

Description

This subroutine changes the type numbers from one set to another one, provided the user has given several sets of type numbers for each problem.

Call

```
CALL CHTYPE ( ISERIES, KPROB )
```

Parameters

INTEGER ISERIES, KPROB(∗)

ISERIES Series number. At output the standard elements have got the $ISERIE^{th}$ series number as read in subroutine PROBDF (4.4. CHTYPE may only be called when more than one series of type numbers has been read in subroutine PROBDF (4.4.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine of subroutine PROBDF (4.4.

Input

ISERIES must have a value.
Array KPROB must have been filled.

Output

The contents of array KPROB have been changed.

4.7 Storing the contents of input arrays into IBUFFR

Description

This subroutine stores the contents of two user provided arrays iinput and rinput in ibuffr/buffer. The storage is such that it can be found for a specific subroutine and sequence number. This subroutine may be used instead of, or additional to the standard input, as read by the starting subroutines like SEPSTN (4.2.3. iinput and rinput may be extended with default values if they are filled partly only.
Mark that FILLINP may be called repeatedly, thus storing all input in IBUFFR.

Heading

```
      subroutine fillinp ( ibuffr, buffer, isubr, iseqnr, iinput,
     +                     rinput, leniinp, lenrinp )
```

Parameters

integer ibuffr(∗), isubr, iseqnr, iinput(∗), leniinp, lenrinp

double precision buffer(∗), rinput(∗)

ibuffr Integer buffer array in which all large data is stored. This is the standard buffer array to be used in SEPRAN. See Section 1.2 for a description of the use of IBUFFR.

buffer Real buffer array in which all real data is stored. Mark that IBUFFR and BUFFER must occupy exactly the same positions. When one uses the blank common BLANK, these arrays are equivalenced.

isubr Indicates the type of subroutine for which the input is meant.
Possible values

1. START 4.3
2. PROBDF 4.4
3. MATSTRUC 4.5
4. CREATE 5.3.1
5. PRESDF 5.2.1
6. FILCOF 6.2.1
7. CHANCF 6.3.2
8. SOLVEL 8.3
9. OUTPUT 20.2.1
10. SEPCOMP (not yet available)
11. DERIVS 14.2
12. NLNPRB 9.2.2
13. INTEGR 13.2.2
14. INTBND 13.3.4
15. TIMPRB 11.2
16. ADAPBOUN 3.9.1
17. ADAPMESH 3.10
18. STATFREE 17.2
19. INSTFREE 17.4
20. CONTACT (not yet available)
21. LOOP (not yet available)
22. CHANGE_COORDINATES (not yet available)

iseqnr Sequence number of input with respect to the specific subroutine. This sequence number is used to refer to later in the call of the actual subroutine.
The minimum value of iseqnr must be 1, the maximum value depends on the specific subroutine and may vary from 10 to 100.

iinput integer input array for the specific subroutine. The user must fill this array himself.
For a description of all these positions, see the specific subroutine to which array IINPUT corresponds.
In case of subroutine SOLVEL, the first position of IINPUT corresponds to the second position of array INPSOL.
In all other cases, the first positions coincide.

rinput Real input array for the specific subroutine The user must fill this array himself.
For a description of all these positions, see the specific subroutine to which array RINPUT corresponds.

leniinp Number of positions of array iinput that have been filled by the user and must be copied to IBUFFR.

lenrinp Number of positions of array rinput that have been filled by the user and must be copied to BUFFER.

Input

The arrays iinput and rinput must have been filled by the user.
The parameters ISEQNR and ISUBR must have been given a value.
The buffer arrays IBUFFR and BUFFER must have been initialized by a SEPRAN starting subroutine.

Output

The contents of the arrays IINPUT and RINPUT are extended with defaults stored in the arrays IBUFFR and BUFFER.

5 Boundary conditions, initial conditions and creation of vectors

5.1 Introduction

In this chapter the various SEPRAN subroutines to define essential boundary conditions, initial conditions and creating SEPRAN vectors are treated. Although these subroutines differ at the outside, internally they are very much the same. In fact an essential boundary condition may also be prescribed by using the creation subroutine. With respect to initial conditions there is no need to a special subroutine, since prescribing an initial condition is exactly the same as creating a solution vector.

In this chapter we deal with the following paragraphs:

5.2 treats the subroutines that may be used to prescribe the essential boundary conditions. Mark that in the "PROBLEM" input, it must already have been defined where essential boundary conditions are available. The various subroutines to prescribe the essential boundary conditions are PRESDF (5.2.1), PRESTM (5.2.2) and PRESBC (5.2.3).

5.3 describes the subroutines that may be used to create a SEPRAN vector (CREATE (5.3.1) and CREATN (5.3.2).

5.4 gives a description of subroutine BVALUE (5.4). This is an old SEPRAN subroutine that may also be used to prescribe essential boundary conditions. The use of BVALUE (5.4) is in practice much more complex than that of the subroutines in Section 5.2. However, one clear advantage is that BVALUE (5.4) does not need any input from the input file.

5.5 deals with subroutine CREATV (5.5) that is used to create SEPRAN vectors without requiring any input from the standard input file. Of course this makes it use much more complex than that of CREATE (5.3.1).

5.6 gives a description of subroutine CREAVC (5.6). This is an old SEPRAN subroutine that may also be used to create vectors.

5.2 Defining the essential boundary conditions: subroutines PRESDF, PRESTM and PRESBC.

In this section we treat the subroutines that may be used to fill the essential boundary conditions in the solution vector. These subroutines all require some kind of input from the user input file.
An much more complex alternative is subroutine BVALUE (5.4) treated in Section 5.4, which does not require any input from the input file.
The standard subroutine is PRESDF (5.2.1) (5.2.1). However, if boundary conditions must be filled in a loop, for example in a time-dependent case, and input can be defined in one call, subroutine PRESTM (5.2.2) forms the natural extension.
The most general of the three subroutines is subroutine PRESBC (5.2.3) (5.2.3). This subroutine not only offers the possibilities of PRESTM (5.2.2), it may also be used in case all input has been read by subroutine SEPSTN (4.2.3).

5.2.1 Subroutine PRESDF

Description

Subroutine PRESDF is used to fill essential boundary conditions in the solution vector, i.e. those boundary conditions that prescribe degrees of freedom. Consult the manual STANDARD PROBLEMS for which variables it concerns.

Call

```
CALL PRESDF ( KMESH, KPROB, ISOL )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), ISOL(5,∗)

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

ISOL array containing information of the solution vector. If more problems are solved in one run of SEPRAN, ISOL should correspond to the solution arrays of all these problems, hence ISOL must have the length NPROB × 5, where NPROB is equal to the number of problems.

Input

The arrays KMESH and KPROB must have been filled.

Furthermore PRESDF requires input from the standard input file. This input is defined in the Users Manual Section 3.2.5, corresponding to the keyword ESSENTIAL BOUNDARY CONDITIONS.
The input is read each time subroutine PRESDF is called.

Output

Essential boundary conditions have been filled into the solution vector.

5.2.2 Subroutine PRESTM

Description

Subroutine PRESTM has exactly the same task as PRESDF (5.2.1). However, PRESTM contains the extra parameter ICTIME, which may be used to indicate if input must be read or if the last input read for this subroutine must be reused.

Call

```
CALL PRESTM ( ICTIME, KMESH, KPROB, ISOL )
```

Parameters

INTEGER ICTIME, KMESH(∗), KPROB(∗), ISOL(5,∗)

ICTIME With the parameter ICTIME the user indicates if the input for subroutine PRESTM should be read from standard input (ICTIME=0), or that the input read by a preceding call of PRESTM must be used (ICTIME=1). If ICTIME = 1, it is necessary that PRESTM has been called before with ICTIME = 0. A call of PRESTM with ICTIME = 0, destroys all input read before.

KMESH, KPROB, ISOL See subroutine PRESDF (5.2.1).

Input

The arrays KMESH and KPROB must have been filled.
ICTIME must have a value.

Furthermore PRESTM requires, depending on ICTIME, exactly the same input from the standard input file as PRESDF (5.2.1).

Output

Essential boundary conditions have been filled into the solution vector.

5.2.3 Subroutine PRESBC

Description

Subroutine PRESBC has exactly the same task as PRESDF (5.2.1) . However, PRESBC may be used to read input from the input file by subroutine SEPSTN (4.2.3), as well as by the subroutine itself. In the first case all input will be read at once, which makes the use more flexible.

Call

```
CALL PRESBC ( KMESH, KPROB, ISOL, IINPRS )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), ISOL(5,∗), IINPRS(∗)

IINPRS Input array defined by the user. With this array the user may indicate which actions he wants to be performed by this subroutine.
IINPRS must be filled as follows:

Pos. 1: Last entry number that has been filled by the user. If 0 or 1 only defaults are used.

Pos. 2: IREAD Defines how the input must be read.
Possible values:

- 0: All SEPRAN input has been read by subroutine SEPSTN (4.2.3) until END_OF_SEPRAN_INPUT or end of file has been found. This input is used.
- 1: The input is read as described for PRESDF (5.2.1).
- 2: The last input read as described for PRESDF (5.2.1) is used (equivalent with PRESTM (5.2.2) ICTIME=1).

Default value: 0

Pos. 3: ISEQNR, This parameter is only used if IREAD=0. It indicates the sequence number of the input for essential boundary conditions to be used.
Default value: 1

Pos. 4: IVECTOR Vector sequence number (first vector)
Default value: 1
If iinprs(1)≥4, iinprs(1) is set equal to the maximum sequence number of the vector created

KMESH, KPROB, ISOL See subroutine PRESDF (5.2.1).

Input

The arrays IINPRS, KMESH and KPROB must have been filled.

Furthermore PRESBC requires, depending on IINPRS, exactly the same input from the standard input file as PRESDF (5.2.1).

Output

Essential boundary conditions have been filled into the solution vector.
Array IINPRS may have been changed.

5.3 Creation of SEPRAN vectors: subroutines CREATE and CREATN

In this section we treat the subroutines that may be used to create SEPRAN vectors which are either of the type solution vector or of the type of vector of special structure. These subroutines all require some kind of input from the user input file.
A much more complex alternative is subroutine CREATV (5.5 treated in Section 5.5, which does not require any input from the input file.
The standard subroutine is CREATE (5.3.1 (5.3.1).
Subroutine CREATN (5.3.2 (5.3.2) is more general than CREATE (5.3.1. This subroutine not only offers the possibilities of CREATE (5.3.1, it may also be used in case all input has been read by subroutine SEPSTN (4.2.3 (Section 4.2).

5.3.1 Subroutine CREATE

Description

Create a SEPRAN vector, and fill it with a constant or function, or refill an existing SEPRAN array.

Call

```
CALL CREATE ( ICTIME, KMESH, KPROB, IVECTR )
```

Parameters

INTEGER ICTIME, KMESH(∗), KPROB(∗), IVECTR(5,∗)

ICTIME Indication if information must be read from the standard input file (0) or that information read before is used (1).
ICTIME = 1 is typically used for time-dependent problems. For the first time step ICTIME should be 0 and input must be read from the standard input file. In the succeeding time-steps usually no input is required and ICTIME = 1 is used. In that case dependencies on time should be introduced by a function subroutine. See input.
A call of CREATE with ICTIME = 0, destroys all input read before.

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4.

IVECTR Array(s) to be filled. Usually IVECTR has the structure of a solution vector, hence ISOL is used as actual argument.
IVECTR may contain one vector, in which case IVECTR is an integer array of length 5. If IVECTR must contain information of n, $(n \geq 1)$ vectors, then IVECTR should be declared as:

INTEGER IVECTR(1:5, 1:n)

IVECTR(.,i) refers to the i^{th} vector, its starting address is given by IVECTR(1,i).

Input

The arrays KMESH and KPROB must have been filled.
Parameter ICTIME must have a value.

Furthermore CREATE requires input from the standard input file. This input is defined in the

Users Manual Section 3.2.10, corresponding to the keyword CREATE.
The input is read each time subroutine CREATE is called.

Output

Array IVECTR has been filled, and the corresponding vector is computed.

5.3.2 Subroutine CREATN

Description

Subroutine CREATN has exactly the same task as CREATE (5.3.1. However, CREATN may be used to read input from the input file by subroutine SEPSTN (4.2.3, as well as by the subroutine itself. In the first case all input will be read at once, which makes the use more flexible.

Call

```
CALL CREATN ( IINCRT, KMESH, KPROB, IVECTR )
```

Parameters

INTEGER IINCRT(∗), KMESH(∗), KPROB(∗), IVECTR(5,∗)

IINCRT Input array defined by the user. With this array the user may indicate which actions he wants to be performed by this subroutine. IINCRT must be filled as follows:

Pos. 1: Last entry number that has been filled by the user. If 0 or 1 only defaults are used

Pos. 2: IREAD Defines how the input must be read Possible values:

- 0: All SEPRAN input has been read by subroutine SEPSTN (4.2.3 until END_OF_SEPRAN_INPUT or end of file has been found. This input is used.
- 1: The input is read as described for CREATE (5.3.1.
- 2: The last input read as described for CREATE (5.3.1 is used (equivalent with CREATE (5.3.1 ICTIME=1).

Default value: 0

Pos. 3: ISEQNR, This parameter is only used if IREAD=0 It indicates the sequence number of the input for essential boundary conditions to be used.
Default value: 1

Pos. 4: IVECTOR Vector sequence number (first vector)
Default value: 1
If iincrt(1)≥4, iincrt(1) is set equal to the maximum sequence number of the vector created

KMESH, KPROB, IVECTR See subroutine CREATE (5.3.1.

Input

The arrays IINCRT, KMESH and KPROB must have been filled.

Furthermore CREATN requires, depending on IINCRT, exactly the same input from the standard input file as CREATE (5.3.1.

Output

Array IVECTR has been filled, and the corresponding vector is computed.
Array IINCRT may have been changed.

5.4 Definition of essential boundary conditions without using the input file: subroutine BVALUE

Description

Essential boundary conditions are filled in the solution vector. BVALUE may be called several times, in order to fill more components of a vector, or to fill boundary conditions on different boundaries. In general, subroutine PRESDF (5.2.1) or CREATE (5.3.1) is preferred above subroutine BVALUE.

Call

```
CALL BVALUE ( ICHOICEBV, ICHBVL, KMESH, KPROB, ISOL, VALUE, ICRV1, ICRV2,
              IDEGFD, JEXCL )
```

Parameters

INTEGER ICHOICEBV, ICHBVL, KMESH(∗), KPROB(∗), ISOL(5,∗), ICRV1, ICRV2, IDEGFD, JEXCL

DOUBLE PRECISION VALUE

ICHOICEBV Choice parameter, possibilities:

0 The degrees of freedom get the constant value of the parameter VALUE.

>0 The values of the degrees of freedom depend on the coordinates. The values are computed using the user written function subroutine FUNCBC. (See SEPRAN INTRODUCTION 5.5.1). The parameter ICHOICEBV is used in FUNCBC as IFUNC.

<0 The degrees of freedom are complex. Independent on whether the values are constant or depend on the co-ordinates, the boundary conditions must be computed with the user written subroutine CFUNCB (See SEPRAN INTRODUCTION 5.5.2). Minus the parameter ICHOICEBV is used in the call of CFUNCB, hence CFUNCB gets a positive value of ICHOICEBV (IFUNC). For complex degrees of freedom ICHOICEBV $\geq$ 0 is not permitted. The parameter VALUE is not used.

>1000 or <-1000 When |ICHOICEBV|>1000 and IDIM=2 or 3 the user written subroutine FUNC1B (ICHOICEBV>1000) or CFUN1B (ICHOICEBV<-1000) is called. See Users Manual 3.3.4.
The parameters VALUE and IDEGFD are then not used.
When IDIM=3 the parameter JSTEP is also not used.
The user must fill the solution vector for each curve element (IDIM=2) or surface element (IDIM=3) separately.

ICHBVL Choice parameter consisting of two parts IDIM and IPROB according to ICHBVL = IDIM + 1000 × (IPROB - 1) with

IDIM Indication on what kind of boundary, boundary conditions must be prescribed. Possibilities:

0 All boundary conditions are set equal to zero. They may get other values by a subsequent call of BVALUE.

1 Boundary conditions are prescribed in user defined points (see subroutine MESH).

2 Boundary conditions are prescribed on curves.

3 Boundary conditions are prescribed on surfaces.

4 Boundary conditions are given in the nodes ICRV1 to ICRV2. JSTEP is not used.

5 Boundary conditions are given in the relative nodal point JSTEP in the elements ICRV1 to ICRV2.

10 + IELGRP Boundary conditions are given in the relative nodal points ICRV1, ICRV1+JSTEP, ICRV1+2×JSTEP, ... ,ICRV2, of all elements of element group IELGRP. When JSTEP=0, JSTEP=1 is assumed. When ICRV1=0 all nodal points in the element group are used.

IPROB The solution vector in which the boundary conditions must be filled corresponds to the problem number IPROB as defined in the PROBLEM input.

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

ISOL In this array information concerning the solution vector will be stored. Boundary conditions are filled in the corresponding solution vector.

ISOL has length 5. In the case that more than one problems must be solved on one mesh ISOL may be declared as a two-dimensional array of length $5 \times n$ (ISOL(5,n)). In the call of BVALUE one can use ISOL(1,$IPROB$)) with IPROB the problem number.

VALUE Value of the constant when the boundary condition is constant.
VALUE is a real parameter, hence a real must be submitted.

ICRV1,ICRV2 Depending on the value of IDIM, all user points with numbers ICRV1, ICRV1 + 1, . . . , ICRV2, all curves with numbers ICRV1, . . . , ICRV2, or all surfaces with numbers ICRV1 to ICRV2 are used. See JSTEP.
For the special cases IDIM>3, see IDIM.

IDEGFD The $IDEGFD^{th}$ degree of freedom in the points as indicated by IDIM, JSTEP, ICRV1 and ICRV2 is used.

JEXCL Choice parameter consisting of two parts: JSTEP and IEXCLUDE according to JEXCL = JSTEP + 100 × IEXCLUDE, with:

JSTEP has the following meaning:

When JSTEP = 0 all user points from ICRV1 to ICRV2 are used when IDIM = 1, all nodal points on the boundaries ICRV1 to ICRV2 when IDIM = 2. When IDIM = 3 and JSTEP = 0 all nodal points on the surfaces ICRV1 to ICRV2 are used.
When IDIM = 2, the curves ICRV1 to ICRV2 must be subsequent curves with coinciding initial and end point, i.e. the end point of ICRV1 must be equal to the initial point of ICRV1 + 1 etc.
When |ICHOICEBV|>1000 and IDIM=2 the value of JSTEP indicates the number of nodal points in each curve element. Hence for a linear element JSTEP=2, for a quadratic element JSTEP=3, etc.

When JSTEP > 0 and IDIM = 1 only the user points ICRV1, ICRV1 + (JSTEP+1), ICRV1 + 2 × (JSTEP+1), . . . are used.
When JSTEP >0 and IDIM =2 only the nodal points 1, 1 + (JSTEP+1), 1 + 2 × (JSTEP+1), . . on the curves ICRV1 to ICRV2 are used.
When JSTEP < 0 and IDIM = 1 all user points between ICRV1 and ICRV2 are used except ICRV1, ICRV1 - (JSTEP-1), ICRV1 - 2 × (JSTEP-1), . . .
When JSTEP < 0 and IDIM = 2 all nodal points on the boundaries ICRV1 to ICRV2 are used except the points 1, 1 - (JSTEP-1), 1 - 2 × (JSTEP-1), . . .

Compare JSTEP with the parameter i in CURVESi of the input ESSBOUNCOND of

subroutine PROBDF (4.4).

When JSTEP > 0 and IDIM = 3, the following possibilities are available:

1 The points 1, 3, 5 in each element, for triangles with shape number 4 or 7 are used; the points 1,4,7 for triangles with shape number 8 ,or the points 1, 3, 5, 7 for quadrilaterals with shape number 6.

2 The points 2, 4, 6 in each element, for triangles with shape numbers 4 or 7 are used, the points 2,3,5,6,8,9 for triangles with shape number 8, or the points 2, 4, 6, 8 for quadrilaterals with shape number 6.

3 Point 7 in each triangle with shape number 7, point 10 for each triangle with shape number 8 is used. Point 9 in each element, for quadrilaterals with shape number 6 are used.

4 Combination of JSTEP = 1 and JSTEP = 3.

5 Combination of JSTEP = 2 and JSTEP = 3.

Other possibilities are not yet available.

IEXCLUDE IEXCLUDE ≠ 0, is only permitted for essential boundary conditions given along curves. In that case the following values for IEXCLUDE are permitted:

0 There are no points to be excluded, except the ones indicated by JSTEP.

1 Besides the points indicated by JSTEP also the initial point of the curve is excluded from being prescribed by BVALUE.

2 Besides the points indicated by JSTEP also the end point of the curve is excluded from being prescribed by BVALUE.

3 Besides the points indicated by JSTEP also the initial and end point of the curve are excluded from being prescribed by BVALUE.

Input

The arrays KMESH and KPROB must have been filled.
ICHOICEBV, ICRV1, ICRV2, VALUE, IDIM, IDEGFD and JEXCL must have a value.

Output

Boundary conditions have been filled in the solution vector.

5.5 Creation of SEPRAN vectors without using the input file: subroutine CREATV

Description

Subroutine CREATV is called by subroutine CREATE (5.3.1), but may also be used separately. CREATV does the actual filling of the SEPRAN arrays.

Heading

```
subroutine creatv ( kmesh, kprob, ivectr, inpcre, rincre )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), IVECTR(5,∗), INPCRE(∗)

DOUBLE PRECISION RINCRE(∗)

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

IVECTR Array(s) to be filled. Usually IVECTR has the structure of a solution vector, hence ISOL is used as actual argument.
IVECTR may contain one vector, in which case IVECTR is an integer array of length 5. If IVECTR must contain information of n, ($n \geq 1$) vectors, then IVECTR should be declared as:

INTEGER IVECTR(1:5, 1:n)

IVECTR(.,i) refers to the i^{th} vector, its starting address is given by IVECTR(1,i).

INPCRE Integer input array in which the user defines how the array(s) IVECTR must be filled.
INPCRE must be filled as follows:

1 Last entry number that has been filled by the user. If INPCRE(1) = 0 or 1 only defaults are used.

2 Number of different vectors in array IVECTR to be created and/or filled.
Default value: 1.

The following positions contain information for each of these vectors sequentially, i.e. first for vector 1, then for vector 2, etc.
Per vector the following information is needed:

1 ICHVC is type of vector. Possible values:
- 0 vector either exists or is a solution vector
- 1 vector is solution vector
- 2 vector is an array of special structure
- 3 vector is an array of special structure defined per element.
- 4 vector of special structure defined per node per element.
- 5 vector of type capacity

Default value: 0.

2 IPROB problem number
Default value: 1.

3 IVEC is only used if ICHVC=2 or 3.
If ICHVC=2, IVEC gives the sequence number of the array of special structure.
If ICHVC=3, IVEC defines the number of degrees of freedom per element.
Default value: 1.

4 ICOMPL indicates if the vector to be created is real (0) or complex (1).
Default value: 0.

5 IFILL indicates how the vector must be filled. Possible values:

-1 the corresponding vector is not filled, only created, i.e. space is reserved in SEPRAN.

0 the corresponding vector is created and made equal to zero.

$m > 0$ the filling of the vector is defined by m commands.

Default value: 0.

Remark: if ICHVC=0 and the vector IVECTR(.,i) exists in the call of subroutine CREATV, then the positions 2, 3 and 4 are not used.

If $m > 0$ information of each of the commands must be given sequentially in the following way:
For each command

First position: indication which degrees of freedom are used. Possible values:

0 All degrees of freedom.

$k > 0$ Only the k^{th} degree of freedom is used.

$k < 0$ $-k$ degrees of freedom are used. The sequence numbers of these degrees of freedom must be stored in the succeeding $-k$ positions.

Next position: indication how these degrees of freedom must be filled. Possible values:

0 Degree of freedom is not filled.

$k < 0$ The degree of freedom is constant.
The value of the constant must be stored in RINCRE($-k$). If the vector is complex RINCRE($-k$) contains the real part of the vector, and RINCRE($-k+1$) the imaginary part.

$0 < k < 1000$ The degree of freedom is a function of the co-ordinates. If the vector is real the value is defined by function subroutine FUNC:
VALUE = FUNC (k, x, y, z)
See SEPRAN INTRODUCTION 5.5.4.

If the vector is complex the vector is filled by the complex function subroutine CFUNC:
VALUE = CFUNC (k, x, y, z), see SEPRAN INTRODUCTION 5.5.4.

$1000 < k < 2000$ Special function with choice number $k-1000$. The degrees of freedom are defined as a function of the co-ordinates, however, in this case the more complicate FUNC1B (real case) and CFUN1B (complex case) must be used:

```
SUBROUTINE FUNC1B(l,INDEX1,INDEX2,USOL,COOR)
```

or

```
SUBROUTINE CFUN1B(l,INDEX1,INDEX2,USOL,COOR)
```

See Users Manual 3.3.4.

$2000 < k < 3000$ Special function with choice number $k - 2000$. The degrees of freedom are defined as a function of the co-ordinates, however, in this case the more complicate FUNCOL (real case) and CFUNOL (complex case) must be used:

```
FUNCTION FUNCOL (k, x, y, z, uold)
```

or

```
FUNCTION CFUNOL (k, x, y, z, uold)
```

See Users Manual 3.3.5.

$3000 < k < 4000$ Special function with choice number $k - 3000$. The degrees of freedom are defined as a function of the co-ordinates and a series of predefined vectors is used.

```
      subroutine funcvect( k, ndim, coor, numnodes, uold,
     +                     nuold, result, nphys)
```

In this case k must be followed by *nuold*, i.e. the number of vectors the new vector depends on followed by the vector sequence numbers.

Next position: Indication in which nodal points the vector must be filled. Possible values:

0 all nodes.

1 only in user points. The next positions contain the number of user points, followed by the user point sequence numbers.

2 only in nodes on the curves C1 to C2, followed by C1, C2 and the parameter i indicating which nodes on the curve must be used. See CREATE (5.3.1) for a definition of this parameter.
The curves C1 to C2 must be subsequent curves, i.e. the end point of C1 must be the initial point of C1+1. This restriction is only valid if $i \neq 0$.
If C1 is equal to -1 then all outer curves are used.

3 only in some nodes in the surfaces S1 to S2, followed by S1, S2, number of nodes per element where the nodes must be filled and then the relative nodal points that must be filled. If number of nodes is set equal to 0 then in all points.
If S1 is equal to -1 then all outer surfaces are used.

4 only in the nodes N_1 to N_2 with step ISTEP followed by N_1, N_2 and ISTEP.

5 only in the elements i to j and relative nodal points
$RN_1, RN_2, ..., RN_k$ followed by
i, j, number of nodes, $RN_1, RN_2, ..., RN_k$.

6 only in some points of the element group *ielgrp* followed by *ielgrp*, number of points and $RN_1 ... RN_k$.

7 only in some nodes in the volumes V_1 to V_2, followed by V_1, V_2, number of nodes per element where the nodes must be filled and then the relative nodal points that must be filled. If number of nodes is set equal to 0, then in all points.

8 only in the user points P_1 to P_2 with step ISTEP followed by P_1, P_2 and ISTEP.

9 only in the active points of obstacle i, followed by i.

10 only in the active points of contact i, followed by i.

11 only in the non-active points of contact i, followed by i.

12 corresponds to the option `FILE_NODAL_VALUES` as described in the Users Manual Section 3.2.10.
The next position in array INPCRE is used. It must contain the position of the name of the file in array TEXTS in common block CSEPCM (21.19).

13 corresponds to the option `FILE_ELEMENT_VALUES` as described in the Users Manual Section 3.2.10.

The next positions in array INPCRE is used. It must contain the position of the name of the file in array TEXTS in common block CSEPCM (21.19).

14 corresponds to the option `FILE_CAPACITY_VALUES` as described in the Users Manual Section 3.2.10.
The next positions in array INPCRE is used. It must contain the position of the name of the file in array TEXTS in common block CSEPCM (21.19).

15 The global unknowns are filled for global group iglgrp.
The next position must contain the global group sequence number.

16 The unknowns corresponding to points in an obstacle are filled.
The next position must contain the obstacle sequence number.

17 The unknowns corresponding to points that correspond only to elements that are completely in an obstacle are filled.
The next position must contain the obstacle sequence number.

18 The unknowns corresponding to points that correspond only to elements that are partly in an obstacle are filled.
The next position must contain the obstacle sequence number.

19 The unknowns corresponding to points that are on the boundary of an obstacle are filled.
The next position must contain the obstacle sequence number.

$k = 9001$ Special boundary condition defining a quadratic function as described in the Users Manual Section 3.2.10 by the option QUADRATIC, MAX = a.
Next position, the position of the value of a in array RINCRE.

RINCRE Double precision input array in which the user puts the real part of the information necessary for the creation of the vector. See INPCRE for a description.

Input

The arrays KMESH and KPROB must have been filled before, for example by subroutine SEPSTR (4.2.1).
The arrays INPCRE and RINCRE must have been filled by the user.

Output

Array IVECTR has been filled, and the corresponding vector is computed.

5.6 Old SEPRAN subroutine for the creation of vectors: subroutine CREAVC

Description

Create a SEPRAN vector, and fill it with a function, or refill an existing SEPRAN array.

Call

```
CALL CREAVC ( ICHOIS, ICHCRV, IVEC, IVECTR, KMESH, KPROB, IU1, U1, IU2, U2 )
```

Parameters

INTEGER ICHOIS, ICHCRV, IVEC, IVECTR(5), KMESH(∗), KPROB(∗), IU1(∗), IU2(∗)

DOUBLE PRECISION U1(∗), U2(∗)

ICHOIS When ICHOIS = 0, the array corresponding to IVECTR is a real array, when ICHOIS = -1, the array is a complex array.

ICHCRV Choice parameter consisting of two parts ICHVC and IPROB according to ICHCRV = ICHVC + 1000 × (IPROB-1) with

ICHVC Type of vector to be created. Possibilities:

1 The vector has the type of the solution vector

2 The vector is a vector of special structure with corresponding sequence number IVEC.

IPROB The array to be created corresponds to the problem number IPROB as defined in the input block PROBLEM.

IVEC When ICHVC = 1, IVEC is not used.
When ICHVC = 2, IVEC gives the sequence number of the vector of special structure.

IVECTR Standard SEPRAN array. In this array information of the function will be stored.
When ICHVC = 1 it has the same structure as the solution array ISOL.
When ICHVC = 2 it is a vector of special structure with sequence number IVEC.
When ICHVC = 3 it is vector defined per element.
When IVECTR(2) has been filled according to the value in the choice parameter ICHVC, then the vector is recognised to be old and the old positions are overwritten, otherwise a new vector is created.
IVECTR has length 5. In the case that more than one problems must be solved on one mesh IVECTR may be declared as a two-dimensional array of length 5 × n (IVECTR(5,n)). In the call of CREAVC one can use IVECTR(1,i) where i refers to the vector one is interested in.

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

IU1,U1 In these arrays of length NUNKP, where NUNKP is the maximal number of degrees of freedom in the nodal points, the user must store information concerning the function in the following way:
IU1(i) corresponds to the i^{th} degree of freedom.
When IU1(i) = 0 the degree of freedom is a constant the value of which must be stored in U1(i).
When IU1(i) > 0 subroutine FUNC (see Introduction manual 5.5.4) is called. The value of IU1(i) gives the parameter ICHOIS in FUNC, hence FUNC is called as:

```
VALUE = FUNC ( IU1(i), X, Y, Z )
```

When the array corresponding to IVECTR is a complex array then IU1(2×i-1) and U1(2×i-1) correspond to the real part of the i^{th} degree of freedom,whereas IU1(2×i) and U1(2×i) correspond to the imaginary part. The length of the arrays IU1 and IU2 must be equal to 2 × NUNKP.
If so-called physical unknowns are introduced, then the length of the arrays IU1 and U1 must be equal to NPHYS in the real case and to 2 × NPHYS in the complex case. NPHYS denotes the number of physical unknowns in a nodal point. Position i in U1 and IU1 then corresponds to the i^{th} physical unknown instead of the i^{th} degree of freedom.

IU2,U2 These parameters are not yet used.

Input

The user must fill the arrays IU1 and U1.
The arrays KMESH and KPROB must have been filled.
ICHOIS and ICHVC must have a value.
When ICHVC = 2 also IVEC must have a value.

Output

Array IVECTR has been filled, and the corresponding vector is computed.

6 Filling of coefficients

6.1 Introduction

In this chapter the various SEPRAN subroutines to define the coefficients for the differential equations and the element subroutines are treated.
If the user utilizes his own elements, he is himself responsible for the filling of the coefficients. Nevertheless also in this case the user may use the subroutines in this chapter.
Coefficients are filled in the arrays IUSER and USER and submitted to the element subroutines. The structure of the arrays IUSER and USER is of no importance if standard elements (type numbers ≥ 100) are used in combination with the SEPRAN fill subroutines. However, if the user uses the filling subroutines of this chapter in combination with his own element subroutines, he needs to know the structure of these arrays. In that case the user has to consult the manual Standard Problems, Section 2.4.
In the case of standard elements it is necessary to give the coefficients in the correct way. Which coefficients are required for a specific problem is treated in the manual Standard Problems.
With respect to the filling of the coefficients there are several possibilities:

- The user may give all input in the input part closed by the statement `end_of_sepran_input`. This makes only sense if the standard program SEPCOMP (or even the subroutine SEPCOM) is used, or if all input is read by the starting subroutine SEPSTN (4.2.3) (Section 4.2).
  This possibility may be used if the user uses matrix building subroutines of a special structure, which do not require that the user submits the arrays IUSER and USER explicitly. For more details about these subroutines consult the Chapters 7, 8 and 9.
  Alternatively this possibility may be used in combination with subroutine FILLCF (6.2.2) (Section 6.2)

- The user may read all input for the coefficients at the moment he needs them. This input is directly stored into the user arrays IUSER and USER. A clear disadvantage of this approach is the sequence dependency of the input. This possibility is provided by subroutine FILCOF (6.2.1) (Section 6.2).

- The user may put the information with respect to the coefficients into the arrays IUSER and USER in the main program. For that reason some help subroutines FILxxx have been developed, which are treated in the manual Standard Problems, Section 2.3.

- The user may fill the arrays IUSER and USER himself. How to do this is described in the manual Standard Problems, Section 2.4.

Besides filling of coefficients the user may also wish to change only a few of the coefficients that have been filled before. Especially for non-linear problems, this option is very useful.
Again the same type of possibilities are open. Special subroutines in this respect are the subroutines CHANCF (6.3.2) and CHNCOF (6.3.1) (Section 6.3).

In this chapter we deal with the following paragraphs:

6.2 treats the subroutines FILCOF (6.2.1) and FILLCF (6.2.2) that may be used to read the information about the coefficients.

6.3 deals with the subroutines CHANCF (6.3.2) and CHNCOF (6.3.1) that may be used to read information about how coefficients may be changed.

6.2 Reading information about the filling of coefficients: subroutines FILCOF and FILLCF.

In this section we treat the subroutines that may be used to read the information about the coefficients and store them into the arrays IUSER and USER.
The standard subroutine is FILCOF (6.2.1) (6.2.1). Each time this subroutine is called input is read from the standard input file.
If the user wants to avoid this reading for each call he may use subroutine FILLCF (6.2.2) (6.2.2). This subroutine not only offers the possibilities of FILCOF (6.2.1), it may also be used in case all input has been read by subroutine SEPSTN (4.2.3) (Section 4.2).

6.2.1 Subroutine FILCOF

Description

Subroutine FILCOF is a help subroutine that may be used to fill coefficients into the arrays IUSER and USER in the case of standard problems. FILCOF reads information of these coefficients from the standard input file.

Call

```
CALL FILCOF ( IUSER, USER, KPROB, KMESH, IPROB )
```

Parameters

INTEGER IUSER(∗), KMESH(∗), KPROB(∗), IPROB

DOUBLE PRECISION USER(∗)

IUSER Integer user array of variable length that is filled with integer information concerning the coefficients of the standard problems. Since IUSER is a variable length array, the first position must be filled by the user with the declared length. In general a length of 500 will be sufficient for standard applications.

USER Double precision user array of variable length that is filled with real information concerning the coefficients of the standard problems. Since USER is a variable length array, the first position must be filled by the user with the declared length. In general a length of 100 will be sufficient for standard applications. However, if the user uses the input option with iref (See input), then the length needed will be considerably longer, since information for each nodal point or each element must be stored.

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

IPROB Problem sequence number.
Remark: for each problem number the user must either use new arrays IUSER and USER, or he must use two-dimensional arrays of the type IUSER($n1$,m), USER($n2$,m) with $n1$ resp. $n2$ the maximal length of IUSER and USER, and m the number of problems. Of course in that case IUSER(1,i), USER(1,i) must be filled by the user for all values of i.

Input

The arrays KMESH and KPROB must have been filled.

The positions 1 and 3 of the arrays IUSER and USER must have been filled.
IPROB must have a value.

Furthermore FILCOF requires input from the standard input file. This input is defined in the Users Manual Section 3.2.6, corresponding to the keyword COEFFICIENTS.
The input is read each time subroutine FILCOF is called.

Output

The arrays IUSER and USER have been filled with information for the subroutines to build matrices and right-hand sides or for the subroutines to compute derivatives.

6.2.2 Subroutine FILLCF

Description

Subroutine FILLCF has exactly the same task as FILCOF (6.2.1). However, FILLCF may be used to read input from the input file by subroutine SEPSTN (4.2.3), as well as by the subroutine itself. In the first case all input will be read at once, which makes the use more flexible.

Call

```
CALL FILLCF ( IUSER, USER, KPROB, KMESH, IINFIL )
```

Parameters

INTEGER IUSER(∗), KMESH(∗), KPROB(∗), IINFIL(∗)

DOUBLE PRECISION USER(∗)

IINFIL Input array defined by the user. With this array the user may indicate which actions he wants to be performed by this subroutine.
IINFIL must be filled as follows:

Pos. 1: Last entry number that has been filled by the user. If 0 or 1 only defaults are used.

Pos. 2: IREAD Defines how the input must be read.
Possible values:

0: All SEPRAN input has been read by subroutine SEPSTN (4.2.3) until END_OF_SEPRAN_INPUT or end of file has been found. This input is used.

1: The input is read as described for FILCOF (6.2.1).

Default value: 0

Pos. 3: ISEQNR, This parameter is only used if IREAD=0. It indicates the sequence number of the input for essential boundary conditions to be used.
Default value: 1

Pos. 4: IPROB, This parameter is only used if IREAD=1. It indicates the problem number
Default value: 1

KMESH, KPROB, IUSER, USER See subroutine FILCOF (6.2.1).

Input

The arrays IINFIL, KMESH and KPROB must have been filled.
The positions 1 and 3 of the arrays IUSER and USER must have been filled.

Furthermore FILLCF requires, depending on IINFIL, exactly the same input from the standard input file as FILCOF (6.2.1).

Output

The arrays IUSER and USER have been filled with information for the subroutines to build matrices and right-hand sides or for the subroutines to compute derivatives.

6.3 Reading information about the changing of coefficients: subroutines CHNCOF and CHANCF.

In this section we treat the subroutines that may be used to read the information about how the coefficients in the arrays IUSER and USER must be changed.
The standard subroutine is CHNCOF (6.3.1) (6.3.1). Each time this subroutine is called input is read from the standard input file.
If the user wants to avoid this reading for each call he may use subroutine CHANCF (6.3.2) (6.3.2). This subroutine not only offers the possibilities of CHNCOF (6.3.1), it may also be used in case all input has been read by subroutine SEPSTN (4.2.3) (Section 4.2).

6.3.1 Subroutine CHNCOF

Description

Subroutine CHNCOF is a help subroutine that may be used to change information about coefficients in the arrays IUSER and USER in the case of standard problems. CHNCOF reads information of these coefficients from the standard input file.
SEPRAN elements is utilized.

Call

```
CALL CHNCOF ( IUSER, USER, KPROB, KMESH, IPROB )
```

Parameters

INTEGER IUSER(∗), KMESH(∗), KPROB(∗), IPROB

DOUBLE PRECISION USER(∗)

IUSER Integer user array of variable length that is filled with integer information concerning the coefficients of the standard problems. Array IUSER must have been filled before by a call to FILCOF (6.2.1) or FILLCF (6.2.2).

USER Double precision user array of variable length that is filled with real information concerning the coefficients of the standard problems. Array USER must have been filled before by a call to FILCOF (6.2.1) or FILLCF (6.2.2).

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

IPROB Problem sequence number.

Input

The arrays KMESH and KPROB must have been filled.
The arrays IUSER and USER must have been filled before by FILCOF (6.2.1) or FILLCF (6.2.2).
IPROB must have a value.

Furthermore CHNCOF requires input from the standard input file. This input is defined in the Users Manual Section 3.2.7, corresponding to the keyword CHANGE COEFFICIENTS.
The input is read each time subroutine CHNCOF is called.

Output

The arrays IUSER and USER have been changed.

6.3.2 Subroutine CHANCF

Description

Subroutine CHANCF has exactly the same task as CHNCOF (6.3.1). However, CHANCF may be used to read input from the input file by subroutine SEPSTN (4.2.3), as well as by the subroutine itself. In the first case all input will be read at once, which makes the use more flexible.

Call

```
CALL CHANCF ( IUSER, USER, KPROB, KMESH, IINFIL )
```

Parameters

INTEGER IUSER(∗), KMESH(∗), KPROB(∗), IINFIL(∗)

DOUBLE PRECISION USER(∗)

IINFIL Input array defined by the user. With this array the user may indicate which actions he wants to be performed by this subroutine.
IINFIL must be filled as follows:

Pos. 1: Last entry number that has been filled by the user. If 0 or 1 only defaults are used.

Pos. 2: IREAD Defines how the input must be read.
Possible values:

0: All SEPRAN input has been read by subroutine SEPSTN (4.2.3) until END_OF_SEPRAN_INPUT or end of file has been found. This input is used.

1: The input is read as described for CHNCOF (6.3.1).

Default value: 0

Pos. 3: ISEQNR, This parameter is only used if IREAD=0. It indicates the sequence number of the input for essential boundary conditions to be used.
Default value: 1

Pos. 4: IPROB, This parameter is only used if IREAD=1. It indicates the problem number
Default value: 1

KMESH, KPROB, IUSER, USER See subroutine CHNCOF (6.3.1).

Input

The arrays IINFIL, KMESH and KPROB must have been filled.
The arrays IUSER and USER must have been filled.

Furthermore CHANCF requires, depending on IINFIL, exactly the same input from the standard input file as CHNCOF (6.3.1).

Output

The arrays IUSER and USER have been changed.

7 Building of the system of equations

7.1 Introduction

In this chapter the various SEPRAN subroutines to build the large matrices and vectors are treated. These subroutines are only necessary if the user controls the complete program himself. In the standard cases they form a part of the integrated solvers like linsol and nlnsol treated in the Chapters 8 and 9.
At this moment there is one subroutine that may be used to create the large matrices and vectors: subroutine BUILD (7.2). This subroutine may be used for standard elements but also for user defined elements as described in the Users Manual Chapter 4.

In this chapter we deal with the following paragraphs:

7.2 treats subroutine BUILD (7.2).

7.3 treats some subroutines that were used in old versions of SEPRAN, i.e. the subroutines SYSTEM (7.3.1), SYSTM0 (7.3.1), SYSTM1 (7.3.2) and SYSTM2 (7.3.3). At this moment there is no need anymore to use one of these subroutines, since there complete task is carried out by BUILD (7.2).
Nevertheless in old SEPRAN programs it may be possible that these subroutines are used. It is guaranteed that these subroutine will exist in the future, unless a conflict with the operating system occurs.

7.2 Subroutine BUILD

Description

Subroutine BUILD may be used to build the large matrix, the large vector and if necessary a diagonal mass matrix. Furthermore the effect of boundary conditions is taken into account.

Call

```
CALL BUILD ( IINBLD, MATR, INTMAT, KMESH, KPROB, IRHSD, MASSMT, ISOL,
             ISLOLD, IUSER, USER )
```

INTEGER IINBLD(∗), MATR(5,∗), INTMAT(5,∗), KMESH(∗), KPROB(∗), IRHSD(5,∗), MASSMT(∗), ISOL(5), ISLOLD(∗), IUSER(∗)

DOUBLE PRECISION USER(∗)

IINBLD In this array it is indicated how the matrix and right-hand side, must be filled.
The entries of IINBLD must be filled (by the user) as follows:

1 Highest entry number of IINBLD that has been filled by the user. The positions 1 to IINBLD(1) must have been given a value by the user. For all other positions default values are used.
If IINBLD(1) = 0 or 1, only defaults are used.

2 JCHOIS general choice parameter, possibilities:

1 Both the large matrix and large vector are built.

2 Only the large vector is built.

3 The large matrix is built and the large vector is made equal to zero. ($\mathbf{f} = 0$).
If EFFBOUNCOND = 0 the large vector may be unequal to zero, due to non-homogeneous essential boundary conditions.

4 The large matrix nor the right-hand-side vector is built.

5 The large matrix is not built; the large vector is made equal to zero. ($\mathbf{f} = 0$).
If EFFBOUNCOND = 0 the large vector may be unequal to zero, due to non-homogeneous essential boundary conditions.

14 The large matrix is built; no right-hand-side vector is made.

Default value: 1

3 EFFBOUNCOND Indication whether the effect of essential boundary conditions must be taken into account (0) or not (1).
Default value: 0

4 IMAS Indication if the mass matrix must be computed (≥ 1) or not (0).
If IMAS = 1 the mass matrix is a diagonal matrix that is stored as a right-hand-side vector, if IMAS = 2 or 3 the mass matrix is a large matrix with the same structure as the "stiffness" matrix MATR.
In the case IMAS=2 the structure of the mass matrix and the stiffness matrix are identical. In the case IMAS=3 array INTMAT must be a two-dimensional array of size 5×2. INTMAT(i,1) refers to the standard large matrix, INTMAT(i,2) refers to the mass matrix.
Default value: 0

5 IPREC Indication if the old solution is stored in ISLOLD (1) or not (0). IPREC is not used if IINBLD(10) > 0.
Default value: 0

6 ALLELEMENTS Indication if all standard element sequence numbers are used (0) or not (1). In that case the positions 16 to 15+NELGRP+NUMNATBND (NELGRP is number of element groups and NUMNATBND the number of boundary element groups)

are considered as an array IELHLP to be filled by the user in the following way:

In IELHLP(IELGRP) (1 ≤IELGRP ≤ NELGRP+NUMNATBND) the user must store, whether elements with standard element sequence number IELGRP (or IELGRP = NELGRP + IBNGRP for boundary elements) are added to the large matrix and large vector or not.
When IELHLP(IELGRP)=0 the elements with standard element sequence number IELGRP are skipped, when IELHLP(IELGRP)=1 these elements are added.
Default value: 0

7 ADDMATVEC When ADDMATVEC = 0 the large matrix and vector are cleared before the assembling process.
When ADDMATVEC = 1 the right-hand-side vector and the large matrix are not cleared, but added to the already existing right-hand-side vector and large matrix.

Hence
RHSD := RHSD + ΔRHSD
MATRIX := MATRIX + ΔMATRIX

Default value: 0

8 CLEARLASTRHSD When CLEARLASTRHSD = 0 the last part of the right-hand-side vector corresponding to the boundary conditions is cleared (standard situation), when CLEARLASTRHSD = 1 the standard assembling process is also applied to this last part of the right-hand-side vector. For most applications CLEARLASTRHSD = 0 suffices.
Default value: 0

9 IPROB Problem definition number. This parameter is used only for historical reasons. If you set IPROB = 0, the default value is used.
Default value: defined by INTMAT.

10 NUMOLD Number of arrays that have been filled in array ISLOLD.
Array ISLOLD may be a dummy array (NUMOLD=0 and IPREC=0), it may contain exactly one vector which has exactly the same structure as the solution vector (NUMOLD=0 and IPREC=1) or ISLOLD may contain information of solution arrays or arrays of special structure. The number of arrays of this type is given by NUMOLD. See array ISLOLD.
If NUMOLD > 0, the value of IPREC is neglected.
Default value: 0.

11 ISPECTRAL Not yet used.

12 NRIGHTHS Number of right-hand sides that must be computed by subroutine BUILD in this call. This parameter is only used if JCHOIS = 1 or 2.
For all these vectors the other parameters in IINBLD are used in exactly the same way.
The information about the right-hand sides is stored in the following way:
f_i refers to rhsd(.,i) . = 1 (1) 5.
Default value: 1.

13 ISEQIN Sequence number of old solution in array ISLOLD.
This option makes only sense if NUMOLD>1, in which case it is not necessary that the old solution vector is the first vector referred to in ISLOLD.
Since for some problems like non-linear ones it is necessary to use the old solution explicitly ISEQIN must contain the sequence number of the old vector.
Default value: 1.

14 IDEFECT_CORRECTION indicates if defect correction should be used.
The defect correction process is described in the Users Manual Section 3.2.8.
If defect correction is used two matrices are made instead of one, and array MATR must be an array of size 5 × 2.
Possible values:

0: no defect correction

1: defect correction

Default value: 0.

15 **ISUBDIVIDE** indicates if quadratic elements must be treated as a cluster of linear elements (1) or as quadratic elements itself (0).
If quadratic elements are subdivided into linear ones the set of linear elements is used to build the matrix. As a consequence the order of approximation is affected.
For example in that case a quadratic triangle is subdivided into 4 linear triangles and 4 times the element subroutine for linear triangles is called instead of one time the one for quadratic triangles.
Of course this option may only be used if the number of degrees of freedom per point is constant.
Default value: 0.

MATR In this array information concerning the large matrices is stored.
Length: $5 \times n$, with n the number of matrices to be built, positions.
In the standard case MATR is an array of size 5 and only one matrix is stored, however, in the case of defect correction two matrices are needed and in that case the size of MATR is 5 $\times$ 2.
MATR(.,1) refers to the matrix to be inverted, which means in the case of defect correction the iteration matrix. This is for example the upwind matrix.
MATR(.,2) refers to the matrix in the right-hand side matrix in (3.2.8.1), (Section 3.2.8 of the User Manual). Usually this is the standard matrix, for example corresponding to a so-called central scheme.
When MATR has been used before, the old positions for the large matrix are used again, hence the preceding large matrix is overwritten. When the user wants to keep the old matrix, he must save it, or use another array MATR.

INTMAT Array containing information of the structure of the large matrix. Output of a SEPRAN starting subroutine (Section 4.2) or subroutine COMMAT (4.5.2).
If IMAS=3 (see IINBLD(4)), INTMAT must have size 5×2, where INTMAT(i,1) refers to the large matrix and INTMAT(i,2) must contain the structure of the mass matrix. If both matrices have the same structure INTMAT(i,2) may be a copy of INTMAT(i,1).

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

IRHSD In this array information concerning the right-hand-side vector will be stored.
Length: $5 \times n$, with n the number of right-hand sides to be built, positions.
The same remarks as for MATR are valid.

MASSMT This array contains the second vector to be created. If IMAS = 1, this matrix is treated as a so-called diagonal mass matrix, which means that it has the same structure as IRHSD, but that no effect of boundary conditions is taken into account. If IMAS >1 it has the same structure as the stiffness matrix.
Array MASSMT is only filled if IMAS >0.

ISOL Array containing information of the solution vector. Essential boundary conditions must be filled in the solution vector corresponding to ISOL if necessary. (Depends on the value of IINBLD). See Section 5.2.
This array is only used to compute the contribution of the essential boundary conditions to the right-hand-side vector. Hence ISOL is only used if essential boundary conditions are present, the right-hand side must be computed and the effect of essential boundary conditions must be taken into account.
Length: 5 positions.

ISLOLD If an iterative procedure is used, for example for non-linear problems the solution in a preceding iteration may be necessary to compute matrices or right-hand side. In that case information concerning the preceding solution must be stored in ISLOLD. The copying of an array ISOL in an array ISLOLD, together with the corresponding solution may be performed by subroutine COPYVC. In the simple case that these matrices and vectors depend only on the solution of a preceding iteration and this solution has exactly the same structure as the solution vector to be computed, it suffices to set IPREC equal to one and NUMOLD equal to zero. ISLOLD corresponds in that case to one solution vector only.

If the matrices and vectors to be created, do not depend on previous solutions or vectors a dummy vector ISLOLD may be used. IPREC and NUMOLD should be both zero for this case.

In the complicated case that the matrices or right-hand side to be built depend on one or more vectors that are computed before, ISLOLD may be a two-dimensional array of length $5 \times k$ (ISLOLD(5,k)), with k at least equal to NUMOLD. Each array ISLOLD(5,i) must refer to an existing (i.e. already computed) solution array or array of special structure. These arrays do not have to correspond to the same problem as long as the problems correspond to the same arrays KMESH and KPROB.

If the user wants to create one array ISLOLD from various arrays ISOL, it is sufficient to copy the 5 positions of each array ISOL into ISLOLD.

Suppose for example that the arrays ISOL(3), IVECT1 and IVECT2 must be placed in the positions 2,3 respectively 1 of ISLOLD, where both ISOL and ISLOLD are two-dimensional arrays. Then the following FORTRAN text may be utilised:

```
          do 100 i = 1,5
             ISLOLD (i,1) = IVECT2(i)
             ISLOLD (i,2) = ISOL(i,3)
             ISLOLD (i,3) = IVECT1(i)
 100       continue
```

In this complicated situation NUMOLD should be larger than zero. The value of IPREC is neglected. Even in the simple situation of one vector of the same structure as ISOL stored in ISLOLD, the user may use NUMOLD=1 and IPREC=0. So this case is exactly the same as NUMOLD=0 and IPREC=1.

USER,IUSER In these arrays information concerning coefficients etc. for the differential equation must be stored. For standard problems that are contained in the library, the arrays IUSER and USER must be filled according to fixed rules (see the manual Standard Problems). Subroutines FILCOF (6.2.1) may be used to fill IUSER and USER. When the user uses his own elements he may fill and use IUSER and USER in his own way, since these arrays are only used by the element subroutines.

Input

The arrays IINBLD, INTMAT, KMESH, KPROB and if necessary IUSER and USER must have been filled.
When there are essential boundary conditions array ISOL must have been filled.
Depending on the value of NUMOLD and IPREC array ISLOLD should be filled.

Output

Depending on the contents of array IINBLD the arrays MATR, MASSMT and IRHSD are filled.

Subroutines called

Subroutine BUILD calls element subroutines. These may be standard elements from the library (corresponding to type numbers larger than 99), or user written element subroutines (corresponding to type numbers smaller than 100). In the Users Manual Sections 4.2-4.4 it is described how user written element subroutines (ELEM, ELEM1, ELEM2) must be programmed. Which of these element subroutines is called by BUILD depends on the parameters NUMOLD and IMAS in that sequence.
If NUMOLD > 0, subroutine ELEM2 is called by subroutine BUILD, else if IMAS > 0, subroutine ELEM1 is called else subroutine ELEM is called.

7.3 Old subroutines for building of the system of equations

In Section 7.2 subroutine BUILD (7.2) has been treated which may be used for the creation of SEPRAN matrices and right-hand-side vectors. In previous versions of SEPRAN some other subroutines have been developed, which take care of some of the tasks that may be carried out by BUILD (7.2). Although not longer recommended, these subroutines still exist and they will be available in future versions of SEPRAN.
In this section we will describe these "old" subroutines.

SYSTEM (7.3.1) Build matrix and right-hand-side vector. This is the basis builder, which allows one matrix and one vector to be built. The solution vector ISLOLD containing results of a preceding iteration must be of the same type as the solution array.
Since a unix command with the same name exists, in most unix environments this routine can not longer be used.

SYSTM0 (7.3.1) Renamed subroutine SYSTEM (7.3.1) in order to avoid conflicts with unix.

SYSTM1 (7.3.2) Subroutine SYSTM1 (7.3.2) is the first extension of SYSTM0 (7.3.1). It is capable of creating one matrix and two vectors at a time.

SYSTM2 (7.3.3) The next extension is subroutine SYSTM2 (7.3.3) which, in addition to the possibilities of SYSTM1 (7.3.2) allows complicated arrays ISLOLD, which may contain more vectors. each of these vectors may be a vector of special structure or a solution vector and may correspond to different problems.

7.3.1 Subroutine SYSTEM, SYSTM0

Description

Subroutine SYSTM0 may be used to build the large matrix and the large vector. Furthermore the effect of boundary conditions is taken into account.

Call

```
CALL SYSTM0 ( ICHOICEBL, MATR, INTMAT, KMESH, KPROB, IRHSD, ISOL, IUSER,
              USER, ISLOLD, IELHLP )
```

INTEGER ICHOICEBL, MATR(∗), INTMAT(5,∗), KMESH(∗), KPROB(∗), IRHSD(∗), ISOL(∗), IUSER(∗), ISLOLD(∗), IELHLP(∗)

DOUBLE PRECISION USER(∗)

ICHOICEBL Choice parameter. ICHOICEBL is equal to: |ICHOICEBL| = JCHOIS + 10 × EFFBOUNCOND + 20 × ALLELEMENTS + 40 × ADDMATVEC + 100 × CLEARLASTRHSD
(When ICHOICEBL<0 array ISLOLD is used also).
The following values for the various parameters are implemented:

JCHOIS general choice parameter, possibilities:

1 Both the large matrix and large vector are built.

2 Only the large vector is built.

3 The large matrix is built and the large vector is made equal to zero. ($\mathbf{f} = 0$).
If EFFBOUNCOND = 0 the large vector may be unequal to zero, due to non-homogeneous essential boundary conditions.

5 The large matrix is not built; the large vector is made equal to zero. ($\mathbf{f} = 0$).
If EFFBOUNCOND = 0 the large vector may be unequal to zero, due to non-homogeneous essential boundary conditions.

14 The large matrix is built; no right-hand-side vector is made.

EFFBOUNCOND Indication whether the effect of essential boundary conditions must be taken into account (0) or not (1).

ALLELEMENTS Indication if all standard element sequence numbers are used (0) or not (1). In that case array IELHLP is used. See IELHLP.

ADDMATVEC When ADDMATVEC = 0 the large matrix and vector are cleared before the assembling process.
When ADDMATVEC = 1 the right-hand-side vector and the large matrix are not cleared, but added to the already existing right-hand-side vector and large matrix.
Hence RHSD := RHSD + ΔRHSD
MATRIX := MATRIX + ΔMATRIX

CLEARLASTRHSD When CLEARLASTRHSD = 0 the last part of the right-hand-side vector corresponding to the boundary conditions is cleared (standard situation), when CLEARLASTRHSD = 1 the standard assembling process is also applied to this last part of the right-hand-side vector. For most applications CLEARLASTRHSD = 0 suffices.

MATR In this array information concerning the large matrix is stored.
Length: 5 positions
When MATR has been used before, the old positions for the large matrix are used again, hence the preceding large matrix is overwritten. When the user wants to keep the old matrix, he must save it, or use another array MATR.

INTMAT Array containing information of the structure of the large matrix. Output of a SEPRAN starting subroutine (Section 4.2) or subroutine COMMAT (4.5).

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

IRHSD In this array information concerning the right-hand-side vector will be stored.
Length: 5 positions

ISOL Array containing information of the solution vector. Essential boundary conditions must be filled in the solution vector corresponding to ISOL if necessary. (Depends on the value of ICHOICEBL). See Section 5.2.
This array is only used to compute the contribution of the essential boundary conditions to the right-hand-side vector. Hence ISOL is only used if essential boundary conditions are present, the right-hand side must be computed and the effect of essential boundary conditions must be taken into account.
Length: 5 positions.

ISLOLD If an iterative procedure is used, for example for non-linear problems the solution in a preceding iteration may be necessary to compute matrices or right-hand side. In that case information concerning the preceding solution must be stored in ISLOLD. If the preceding solution is not necessary, a dummy parameter may be used.
ISLOLD is only used if ICHOICEBL<0.
The copying of an array ISOL in an array ISLOLD, together with the corresponding solution may be performed by subroutine COPYVC (12.5).

USER,IUSER In these arrays information concerning coefficients etc. for the differential equation must be stored. For standard problems that are contained in the library, the arrays IUSER and USER must be filled according to fixed rules (see the manual Standard Problems). Subroutines FILCOF (6.2.1) may be used to fill IUSER and USER. When the user uses his own elements he may fill and use IUSER and USER in his own way, since these arrays are only used by the element subroutines.

IELHLP Help array to be used when ALLELEMENTS = 1.
In IELHLP(IELGRP + 5) ($1 \leq$ IELGRP $\leq$ NELGRP+NUMNATBND) the user must store, whether elements with standard element sequence number IELGRP (or IELGRP = NELGRP + IBNGRP for boundary elements) are added to the large matrix and large vector or not.
When IELHLP(IELGRP+5) = 0 the elements with standard element sequence number IELGRP are skipped, when IELHLP(IELGRP+5) = 1 these elements are added.
IELHLP is a variable length array, hence position 1 must be filled by the user.

Input

The arrays INTMAT, KMESH, KPROB and if necessary IUSER and USER must have been filled.
When there are essential boundary conditions array ISOL must have been filled.
ICHOICEBL must have a value
When ICHOICEBL $<$ 0 array ISLOLD must have been filled.
Depending on the value of ICHOICEBL array IELHLP must have been filled.

Output

Depending on ICHOICEBL the arrays MATR and IRHSD are filled.

Subroutines called

Subroutine SYSTM0 calls element subroutines. These may be standard elements from the library (corresponding to type numbers larger than 99), or the user written element subroutine ELEM (corresponding to type numbers smaller than 100). See the Users Manual Section 4.2.

7.3.2 Subroutine SYSTM1

Description

Subroutine SYSTM1 may be used to build the large matrix and the large vector as well as a diagonal mass matrix. Furthermore the effect of boundary conditions is taken into account.

Call

```
CALL SYSTM1 ( ICHOICEBL, IMAS, MATR, INTMAT, KMESH, KPROB, IRHSD, MASSMT,
              ISOL, IUSER, USER, ISLOLD, IELHLP )
```

INTEGER ICHOICEBL, IMAS, MATR(∗), INTMAT(5,∗), KMESH(∗), KPROB(∗), IRHSD(∗), MASSMT(∗), ISOL(∗), IUSER(∗), ISLOLD(∗), IELHLP(∗)

DOUBLE PRECISION USER(∗)

ICHOICEBL, MATR, INTMAT, KMESH, KPROB See subroutine SYSTM0 (7.3.1) (7.3.1).

IRHSD, ISOL, IUSER, USER, ISLOLD, IELHLP See subroutine SYSTM0 (7.3.1) (7.3.1).

IMAS Choice parameter for the vector corresponding to MASSMT. Possibilities:

0 Array MASSMT is not created.

1 Array MASSMT is created.

MASSMT This array contains the second vector to be built. In general this array is treated as a so-called diagonal mass matrix, which means that it has the same structure as IRHSD, but that no effect of boundary conditions is taken into account.

Input

The arrays INTMAT, KMESH, KPROB and if necessary IUSER and USER must have been filled.
When there are essential boundary conditions array ISOL must have been filled.
ICHOICEBL and IMAS must have a value
When ICHOICEBL < 0 array ISLOLD must have been filled.
Depending on the value of ICHOICEBL array IELHLP must have been filled.

Output

Depending on ICHOICEBL and IMAS the arrays MATR, MASSMT and IRHSD are filled.

Subroutines called

Subroutine SYSTM1 calls element subroutines. These may be standard elements from the library (corresponding to type numbers larger than 99), or the user written element subroutine ELEM1 (corresponding to type numbers smaller than 100). See the Users Manual Section 4.3.

7.3.3 Subroutine SYSTM2

Description

Subroutine SYSTM2 may be used to build the large matrix and the large vector as well as a diagonal mass matrix. Furthermore the effect of boundary conditions is taken into account.

Call

```
CALL SYSTM2 ( ICHOICEBL, IMAS, IPROB, MATR, INTMAT, KMESH, KPROB, IRHSD,
              MASSMT, ISOL, IUSER, USER, NUMOLD, IVCOLD, IELHLP )
```

INTEGER ICHOICEBL, IMAS, IPROB, MATR(∗), INTMAT(5,∗), KMESH(∗), KPROB(∗), IRHSD(∗), MASSMT(∗), ISOL(∗), IUSER(∗), NUMOLD, IVCOLD(∗), IELHLP(∗)

DOUBLE PRECISION USER(∗)

ICHOICEBL, IMAS, KMESH, KPROB, IUSER, USER See subroutine SYSTM1 (7.3.2) (7.3.2).

IELHLP, INTMAT, IRHSD, MASSMT, ISOL See subroutine SYSTM1 (7.3.2) (7.3.2).

MATR See subroutine SYSTM1 (7.3.2) (7.3.2).
With respect to more problems to be solved on one mesh the following remark is valid for all these arrays.
Each of these arrays has length 5. In the case of more problems to be solved on the same mesh, one may declare these arrays as two-dimensional arrays of length $5 \times n$, for example MATR(5,n). In the call of SYSTM2 one can then use ARRAY(1,IPROB), for example MATR(1,IPROB).

IPROB Problem number of the problem to be solved. This number relates to the problem numbers such as they are read by subroutine SEPSTR (4.2.1) or PROBDF (4.4).

IVCOLD Two-dimensional array of length $5 \times k$ (IVCOLD(5,k)), with k at least equal to NUMOLD. Each array IVCOLD(5,i) must refer to an existing (i.e. already computed) solution array or array of special structure. These arrays do not have to correspond to the same problem as long as the problems correspond to the same arrays KMESH and KPROB.
If the user wants to create one array IVCOLD from various arrays ISOL, it is sufficient to copy the 5 positions of each array ISOL into IVCOLD.
Suppose for example that the arrays ISOL(3), IVECT1 and IVECT2 must be placed in the positions 2,3 resp 1 of IVCOLD, where both ISOL and IVCOLD are two-dimensional arrays. Then the following FORTRAN text may be utilized:

```
          do 100 i = 1,5
             IVCOLD (i,1) = IVECT2(i)
             IVCOLD (i,2) = ISOL(i,3)
             IVCOLD (i,3) = IVECT1(i)
 100      continue
```

Input

The arrays INTMAT, KMESH, KPROB, IVCOLD and if necessary IUSER and USER must have been filled.
When there are essential boundary conditions array ISOL must have been filled.
ICHOICEBL, IPROB and IMAS must have a value
Depending on the value of ICHOICEBL array IELHLP must have been filled.

Output

Depending on ICHOICEBL and IMAS the arrays MATR, MASSMT and IRHSD are filled.

Subroutines called

Subroutine SYSTM2 calls element subroutines. These may be standard elements from the library (corresponding to type numbers larger than 99), or the user written element subroutine ELEM2 (corresponding to type numbers smaller than 100). See the Users Manual Section 4.4.

8 Solution of linear equations

8.1 Introduction

In this chapter the various SEPRAN subroutines to solve linear systems of equations constructed by the subroutines in Chapter 7 are treated. Also subroutines that combine the creation of the system of equations with the solution of the linear system are considered.
The linear solvers itself are only necessary if the user controls the complete program himself. In the standard cases they form a part of the integrated solvers like linsol and nlnsol treated in this chapter and Chapter 9.
There are a number of subroutines that may be used to create and solve a system of linear equations in one call.
Also there are a number of subroutines that may be used to solve a system of linear equations created separately by subroutines treated in Chapter 7.

In this chapter we deal with the following paragraphs:

8.2 treats subroutines that are used to create and solve the systems of linear equations in one call. It concerns the subroutines LINSOL (8.2.1), LINPRB (8.2.3) and LINSTM (8.2.2).

8.3 treats the standard linear solver. This solver (SOLVEL (8.3)) contains all possible subsolvers.

8.4 treats the available direct linear solver (SOLVE). In fact this solver is also called by subroutine SOLVEL (8.3), but in old SEPRAN programs the explicit call of SOLVE may be present.

8.5 treats the standard iterative CG-type solvers. These solvers are in fact also called by subroutine SOLVEL (8.3). This section is only available because of historical reasons.

8.6 deals with the linear solvers based on overrelaxation. This implies also the solution of constrained optimization problems. These solvers too are called by SOLVEL (8.3).

8.2 Subroutines to build and solve linear equations in one call

In this section we consider the various SEPRAN subroutines that may be used to build and solve the system of equations in one call.
These subroutines perform the following tasks:

- Reading and defining of the coefficients.
- Building of the system of equations and the right-hand side.
- Solution of the resulting system of equations.

In fact the call of LINSOL (8.2.1) as described in Section 2.1 may be replaced by three other calls:

```
      call filcof ( iuser, user, kprob, kmesh, iprob )
      call build ( iinbld, matr, intmat, kmesh, kprob, irhsd, massmt, isol,
     +             islold, iuser, user )
      call solve ( iposst, matr, isol, irhsd, intmat, kprob)
```

At this moment there are three subroutines available:

LINSOL (8.2.1) (8.2.1) is the oldest and simplest of the three. It performs exactly the task described before.

LINSTM (8.2.2) is the natural extension to be used in for example time-dependent or perhaps non-linear cases. Input is read only once.

LINPRB (8.2.3) is the most sophisticated of the subroutines. It does not only perform the same tasks as LINSOL (8.2.1), it is also able to deal with input read by subroutine SEPSTN (4.2.3). This input concerns both the coefficients as well as information about the linear solver.

8.2.1 Subroutine LINSOL

Description

Subroutine LINSOL is the typical solution subroutine for linear problems. The subroutine performs the following tasks:

- Reading and definition of coefficients for the problems to be solved.
- Building of the system of equations and the right-hand side.
- Solution of the problem.
  The most important difference with the sequence of subroutines FILCOF (6.2.1), BUILD (7.2) and SOLVEL (8.3) is that LINSOL (8.2.1) requires input from the standard input file with respect to the solution subroutine, and as a consequence is therefore easier to handle. In time-dependent problems, however, input from the standard input file may be useful, if for each time-step input is required. To overcome this problem an extension of LINSOL (8.2.1): LINSTM (8.2.2) is made, which requires only input from the standard input file at times the users wishes.
- For a limited class of linear partial differential equations LINSOL (8.2.1) is able to solve the problem with prescribed accuracy by using an adaptive mesh refinement.
  See the manual Standard Problems for a description of the problems that can be used in combination with adaptive mesh refinement.

Call

```
CALL LINSOL ( KMESH, KPROB, INTMAT, ISOL )
```

INTEGER KMESH(∗), INTMAT(5), KPROB(∗), ISOL(∗)

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

INTMAT Array containing information of the structure of the large matrix. Output of a SEPRAN starting subroutine (Section 4.2) or subroutine COMMAT (4.5.2).

ISOL Array containing information of the solution vector. If there are any essential boundary conditions they must have been filled before in the solution vector corresponding to ISOL. At output the solution of the problem is stored in the vector corresponding to ISOL. Length: 5 positions.

Input

The arrays INTMAT, KMESH, KPROB must have been filled.
When there are essential boundary conditions array ISOL must have been filled.

Furthermore subroutine LINSOL (8.2.1) requires input from the standard input file.
This input consists of the parts corresponding to the keywords COEFFICIENTS and SOLVE as described in the Users Manual Sections 3.2.6 and 3.2.8 in that sequence.
Each time LINSOL (8.2.1) is called this input is read.

Output

The solution vector corresponding to ISOL has been filled.

Subroutines called

Subroutine LINSOL (8.2.1) calls among others, subroutine BUILD (7.2), and for that reason all element subroutines connected with it. See Section 7.2.

8.2.2 Subroutine LINSTM

Description

Subroutine LINSTM (8.2.2) is a simple extension of LINSOL (8.2.1) (8.2.1) especially meant for time-dependent problems. It requires only input from the standard input file the first time it is called.

Call

```
CALL LINSTM ( ICTIME, KMESH, KPROB, INTMAT, ISOL )
```

INTEGER ICTIME, KMESH(∗), INTMAT(5), KPROB(∗), ISOL(∗)

KMESH, KPROB, INTMAT, ISOL See subroutine LINSOL (8.2.1).

ICTIME With the parameter ICTIME the user indicates if the input for subroutine LINSTM (8.2.2) should be read from standard input (ICTIME=0), or that the input read by a preceding call of LINSTM (8.2.2) must be used (ICTIME=1). If ICTIME = 1, it is necessary that LINSTM (8.2.2) has been called before with ICTIME = 0. A call of LINSTM (8.2.2) with ICTIME = 0, destroys all input read before.

Input

The arrays INTMAT, KMESH, KPROB must have been filled.
When there are essential boundary conditions array ISOL must have been filled.
ICTIME must have a value.

If ICTIME=0, LINSTM (8.2.2) requires exactly the same input from the standard input file as LINSOL (8.2.1).

Output

The solution vector corresponding to ISOL has been filled.

8.2.3 Subroutine LINPRB

Description

Subroutine LINPRB is the most general of the three subroutines. It is able to read all input from the standard input file, just as in the case of LINSOL (8.2.1) or LINSTM (8.2.2), but is also able to interpret the input read by SEPSTN (4.2.3).

Call

```
CALL LINPRB ( KMESH, KPROB, INTMAT, ISOL, IINSOL )
```

INTEGER KMESH(∗), INTMAT(5), KPROB(∗), ISOL(5,∗), IINSOL(∗)

KMESH, KPROB, INTMAT See subroutine LINSOL (8.2.1).

ISOL Array containing information about the solution vector and possibly extra vectors needed for the building of the system of equations. It is not necessary that the solution vector is stored in ISOL(.,1), however, if the solution vector corresponds to one of the other vectors stored in ISOL, it is necessary to use IINSOL(6)

IINSOL Input array defined by the user. With this array the user may indicate which actions he wants to be performed by this subroutine.
IINSOL must be filled as follows:

Pos. 1: Last entry number that has been filled by the user. If 0 or 1 only defaults are used.

Pos. 2: IREAD Defines how the input must be read.
Possible values:

- 0: All SEPRAN input has been read by subroutine SEPSTN (4.2.3) until END_OF_SEPRAN_INPUT or end of file has been found. This input is used.
- 1: The input is read as described for LINSOL (8.2.1).
- 2: The last input read as described for LINSOL (8.2.1). is used (equivalent with LINSTM (8.2.2) ICTIME=1)

Default value: 0

Pos. 3: ISEQCF, This parameter is only used if IREAD=0.
It indicates the sequence number of the input for the coefficients to be used.
Default value: 1

Pos. 4: ISEQSL, This parameter is only used if IREAD=0.
It indicates the sequence number of the input for the linear solver to be used.
Default value: 1

Pos. 5: IPROB, Problem sequence number.
Default value: 1

Pos. 6: IVECTR, sequence number of solution vector with respect to array ISOL.
Default value: iprob

Pos. 7: MAXVEC, Maximum number of vectors stored in ISOL.
Default value: 1

Pos. 8: IDEF_COR indicates if defect correction should be used.
Possible values:

- 0: no defect correction
- 1: defect correction

Default value: 0

Pos. 9: LINEAR_SUB indicates if, with respect to the building of the matrices and right-hand sides, quadratic elements should be treated as a cluster of linear subelements. See Section 7.2 for the details.

0: no subelements

1: subelements

Default value: 0

Pos.10: IVEC_REACT indicates if reaction forces must be computed (>0) or not (0).
If > 0, IVEC_REACT defines the sequence number of the vector in which the reaction forces must be stored.
Default value: 0

Pos.11: IVEC_FEM is only used in case of a spectral mesh. It indicates if also the fem solution vector. must be stored besides the sem solution vector. if > 0, IVEC_FEM defines the sequence number of the fem solution vector
Default value: 0

Pos.12: Indicates if a FEM preconditioning must be applied in case of a spectral mesh.
Possible values:

0: no fem preconditioning

1: fem preconditioning

Default value: 0

Pos.13: KEEP indicates if the matrix and right-hand side must be kept, destroyed or reused.
Possible values:

0: new and destroy

1: new and keep

2: old and destroy

3: old and keep

Default value: 0

Pos.14: IVEC_REACT_SP indicates if reaction forces must be computed (>0) or not (0) The difference with 10 is that subroutine MAVER (12.8) is called with 9 instead of 10. This implies that the vector must have been filled before and moreover that matrix times solution without the effect of essential boundary conditions are stored. The rest of the vector is unchanged.
If >0, IVEC_REACT_SP defines the sequence number of the vector in which the reaction forces must be stored.
Default value: 0

Input

The arrays INTMAT, KMESH, KPROB and IINSOL must have been filled.
When there are essential boundary conditions array ISOL must have been filled. Vectors needed in BUILD (7.2) must have been filled before in ISOL.

Depending on the parameter IREAD, LINPRB (8.2.3) may need input from the standard input file. This input is exactly the same as for subroutine LINSOL (8.2.1).

Output

The solution vector corresponding to ISOL has been filled.

8.3 General subroutine for the solution of linear equations previously created (SOLVEL)

Description

Subroutine SOLVEL is the standard SEPRAN linear solver. This solver is, at the users choice, able to solve systems of linear equations created by SEPRAN by either direct methods or by iterative methods. Also linear equations with constraints may be solved. SOLVEL itself calls the subroutines that are treated in the next subsections, and hence there is no need for the use of these subroutines separately.

Heading

```
      subroutine solvel ( inpsol, rinsol, matr, isol, irhsd, intmat,
     +                    kmesh, kprob, iread )
```

Parameters

INTEGER INPSOL(∗), MATR(5,∗), ISOL(5), IRHSD(5), INTMAT(5), KMESH(∗), KPROB(∗), IREAD

DOUBLE PRECISION RINSOL(∗)

INPSOL Input array defined by the user. With this array the user may indicate which actions he wants to be performed by this subroutine.
INPSOL must be filled as follows:

Pos. 1: Last entry number that has been filled by the user. If 0 or 1 only defaults are used.

Pos. 2: IPOS Is used only for direct methods.
Indicates if the matrix to be solved is positive definite, (IPOS = 1) or not (IPOS = 0).
Default: 0

Pos. 3: SOLUTION_METHOD Indicates the type of solver used.
Possible values:

-1: activates a very special direct solver based on a compact storage. To use this solver it is necessary to define the matrix storage with METHOD = 6. This direct solver is in fact the method Y12M of Zlatev (19). This method can only be used if the Y12M solver is available in your institute. Y12M is a direct solver, which uses pivoting. However, it is based on a compact storage. Extra space needed for the Gaussian elimination process is created during the process itself. For large systems of equations the storage required may be less than for the profile solver. At this moment the use of Y12M is not recommended unless the system of equations can only be solved if pivoting is applied.

0: A profile method is used (direct solver).

1: The system of equations is solved by an iterative solver of conjugate gradients type. If the matrix is symmetrical CG (conjugate gradients) is applied, otherwise BiCGstab. (Sonneveld and van der Vorst)

2: The conjugate gradients squared method (CGS) is used. (Sonneveld)

3: The so-called GMRES method with restart (Saad) is used.

4: The GMRESR solver is used (v.d.Vorst and Vuik).
This is a type of GMRES method with as inner loop the GMRES method and as outer loop the GCR method using a variable polynomial preconditioner. If the number of iterations for GMRES is large and a super linear convergence is visible, GMRESR may improve the convergence speed compared to GMRES.

5: The system of equations is solved by an overrelaxation method. However, for this type of iteration, METHOD may not be equal to 5 or 6, but must be equal to 9.
6: The SIMPLE_GCR method is used for (Navier-)Stokes type equations.
7: The SYMMLQ method is used for symmetric indefinite problems.
8: The SIMPLER_GCR is used.
9: ML_multigrid.
10: IDR.
11: AL_GCR.
12: HSIMPLER_GCR.
13: MSIMPLER_GCR.
14: LSC_GCR.
15: QLSC_GCR.
16: SCHUR.
17: Bi_CGSTab(l).
18: BLOCK_TRIANGULAR
Default value: 0

Pos. 4: IPRECO Defines the type of preconditioning in case of iterative methods.
Possible values:
0: No preconditioning.
1: The matrix is premultiplied by a diagonal matrix, such that all diagonal elements become 1. (Diagonal scaling).
2: Incomplete LU-decomposition using the diagonal as only changed elements. Efficient Eisenstat implementation.
3: Incomplete LU-decomposition, including shifted Laplace.
4: Symmetric Gauss-Seidel.
5: Incomplete LU-decomposition using a pre- assembled decomposition according to a method of Rik Kaasschieter.
6: a standard (=default) preconditioner,
7: See 2, however, MILU (modified eisenstat) is applied.
Default value: 6

In case of Y12M IPRECO defines lu_storage factor n, i.e. the amount of storage that must be defined for the LU-decomposition. n defines the multiplication factor, i.e. the space used for the LU-decomposition is $n \times$ the storage needed for the original matrix. If n is too large, possibly no space is available. If n is too small the process may become very slow or even may terminate without finding the solution.
Default value: 15

Pos. 5: MAXITR Maximum number of iterations to be performed.
In case of a direct solver iterative improvement is applied if maxitr > 0, with at most maxitr iterations.
Default: 0 (implies number of unknowns)

Pos. 6: IPRINT Print level of subroutine
Possible values:
-1 No print output
0 Only error messages are printed
1 A little amount of information of the iteration method is printed.
2 A maximal amount of information of the iteration method is printed.
3 See 2, in this case also the matrix and right-hand side are printed.
Default: 0

Pos. 7: MATSYM Is used only when a nonsymmetric storage scheme METHOD=9,10,11,12 is used. It tells that the matrix is symmetric, so the CG methods can be used.
Possible values:

0: The matrix is supposed to be nonsymmetric

1: The matrix is symmetric.

Pos. 8: MGMRES Dimension of Krylov space for GMRES.
Default: 20
In case of the solver Y12M it indicates the number of pivot rows that are used (Default 3)

Pos. 9: ISTART Control parameter concerning the start of the conjugate process. (ISTART).
Possibilities:

0: The process starts with vector 0

1: The process starts with an explicitly given start vector.

2: The process starts with a random vector.

Default: 0

Pos. 10: KEEP With this parameter the user may indicate whether the preconditioning matrix must be destroyed, kept or that an old preconditioning matrix must be used.
Possibilities:

0 The preconditioning matrix is destroyed.

1 The preconditioning matrix is kept. A reference is written in MATR(4).

2 The preconditioning matrix as computed before is used. It is supposed that the reference for this matrix has been stored in MATR(4).

Default: 0

Pos. 11: IRELER: Indicates which criterion should be used for the termination of the process.
Consists of two parts: IRELER_orig and IABS according to IRELER = IRELER_orig + 100 × IABS.
Possible values for IRELER:

-1: The relative error is taken with respect to $\| \mathbf{b} \|$.

0: The relative error is taken with respect to $\| \mathbf{x} \|$.

1: The relative error is taken with respect to $\| \mathbf{res_0} \|$, where $\mathbf{res_0}$ is the initial residual.

2: The absolute error is taken.

10: Use, default value, i.e. 1 except for CG where 0 is used

Possible values for IABS:

0: No special action

1: Besides the error using IRELER_orig it is also checked if the absolute error is smaller than EPS2 (RINSOL(7))

Default: IRELER = 10

Pos. 12: IRESID: Indicates whether the residual must be computed and printed after computation of the solution.
Possible values:

0: Do not compute the residual.

1: Compute and print the 2-norm of the residual.

2: Compute and print the 2-norm of the residual divided by the 2-norm of the right-hand side.

Default: 0

Pos. 13: NTRUNC: is used in combination with GMRESR or CG for a symmetric matrix.
In combination with GMRESR it defines the maximum number of search directions. NTRUNC should be much smaller than MGMRES.
In combination with CG for a symmetric matrix it defines the maximum number of iterations to estimate the smallest eigenvalue. This eigenvalue is used to correct the termination criterion for the condition of the matrix. Once the number of iterations equals NTRUNC the then computed eigenvalue is used for the rest of the process. If the matrix is very ill-conditioned it may be necessary to choose NTRUNC larger than the default value.
Default: 20

Pos. 14: NINNER: This parameter is only used if GMRESR is used. It indicates the maximum number of iterations in the inner loop of the GMRESR process.
Default: 5
In case of IDR, it contains the parameter s of IDR(s) and in case of BiCGSTAB(l): l.
Default: 1

Pos. 15: NITER1: This parameter is only used if overrelaxation is applied. See the section "How to influence the overrelaxation process" in Section 3.2.8 of the Users Manual.
NITER1 is the variable in common CITER (Section 8.5). If INPSOL(15) = -3, NITER1 is set equal to 0, if INPSOL(15) = 0, NITER1 is not changed.
Default: 0
In the case of CG it indicates if a minimum eigenvalue is given that is used in the termination test (1) or not (0).
Default: 0

Pos. 16: NITER2: This parameter is only used if overrelaxation is applied. NITER2 is the variable in common CITER. If INPSOL(16) = 0, NITER2 is not changed.
Default: 0
In the case of CG it indicates PROJ_PRINTLEVEL: i.e. the amount of output produced when computing the projection vectors. Possible levels: -1, 0, 1, 2, where -1 means no output, 0 standard output and > 0 extra output.
Default: 0

Pos. 17: ICONSTRAIN: This parameter is only used if overrelaxation is applied. ICONSTRAIN indicates if constraints should be applied.
Possible values:

- 0: No constraints, standard overrelaxation (subroutine OVERRL (8.6.1)) is applied
- 1: The solution must satisfy $u \geq min(x)$
- 2: The solution must satisfy $u \leq max(x)$
- 3: The solution must satisfy $min(x) \leq u \leq max(x)$

If ICONSTRAIN > 0, overrelaxation with constraints (subroutine OVERCS (8.6.2)) is applied.
Default: 0

Pos. 18: IMINIMUM: This parameter is only used if constrained overrelaxation is applied. It indicates how the minimum must be computed.
Possible values:

- 0: min(x) = 0
- >0: min(x) is a constant (rinsol(7))
- <0: min(x) is a function: FUNC(-IMINIMUM,...)

Default: 0
In case of CG this position is used to define the projection method.
Possible values:

0: none
1: projection based on approximate eigenvectors
Default: 0

Pos. 19: IMAXIMUM: This parameter is only used if constrained overrelaxation is applied. It indicates how the maximum must be computed.
Possible values:
0: max(x) = 0
>0: max(x) is a constant (rinsol(8))
<0: max(x) is a function: FUNC(-IMAXIMUM,...)
Default: 0
In the case of CG it indicates PROJ_PLOTVECTORS which indicates if the projection vectors must be plotted
Possible values:
0: do not plot
1: make a contour plot
Default: 0

Pos. 20: IDEGFD: This parameter is only used if constrained overrelaxation is applied. It indicates the the degree of freedom that must be restricted. If 0 all are restricted.
Default: 0
In the case of CG it indicates PROJ_TYPE which indicates how the projection vectors must be computed.
Possible values:
1: by solving a Poisson equation for the shale layers
2: by setting shale layers equal to 1
3: by using a 0/1 projection vector for each layer
Default: 1

Pos. 21: NUMDEF_CORR: Number of defect correction steps to be performed. For details consult the Users Manual Section 3.2.8.
Default: 0

Pos. 22: AT_ERROR: defines what to do in case of an error. Possible values:
0 stop
1 return to calling program

Pos. 23: ISEQ_EXACT: Sequence number of exact solution in ISOL.
If zero the exact solution is not given.
Default: 0

Pos. 24: ISEQ_RES_EXACT: Sequence number of the residual of exact solution in ISOL.
If zero the residual is not computed.
This position can only be used in combination with ISEQ_EXACT.
Default: 0

Pos. 25: NPAIRS Number of pairs where the special boundary conditions for the over-pressures must be given.
Default: 0

Pos. 26: IPOSPAIRS Is only used if NPAIRS>0 |IPOSPAIRS| defines the starting address of array PAIRS in INPSOL.
If IPOSPAIRS < 0, user points are given, if IPOSPAIRS> 0 node numbers.
This starting address must be at least equal to 41.
Default: 0

Pos. 27: RPOSPAIRS is only used if NPAIRS>0.
Defines the start of the threshold values in RINSOL.
Must be at least equal to 11.
Default: 0

Pos. 28: ISEQ_START_RES Contains the position of start residual in array ISOL.
Default: 0

Pos. 29: ISEQ_END_RES Contains the position of end residual in array ISOL.
Default: 0

Pos. 30: ISEQ_DIAG Contains the position of diagonal of matrix in array ISOL.
Default: 0

Pos. 31: ISEQ_PREC_DIAG Contains the position of diagonal of preconditionings matrix in array ISOL.
Default: 0

Pos. 32: ILIMIT defines if the solution must be limited to the minimum and maximum values in the input vector, including boundary conditions.
Possible values:

0 no limiting

1 limiting

Default: 0

Pos. 33: PRLUMP defines if the matrix must be lumped before the preconditioning matrix is computed (1) or not (0).
See the Users Manual Section 3.2.8 for a description.
Default: 0, except when the preconditioning matrix does not exist.

Pos. 34: ISCALING defines the type of scaling.
Possible values:

0 none

1 row scaling

2 symmetric scaling

3 column scaling

See the Users Manual Section 3.2.8 for a description.
Default: 0

Pos. 35: NUMSUBEQ defines the number of subequations.

Pos. 36: NDEFLAT defines the type of deflation method to be applied.
Possible values:

0 deflation

1 coarse grid correction

See the Users Manual Section 3.2.8 for a description.

Pos. 37: IPRINT_SPECTRAL Print level of main subroutine for spectral solution. This is not actually used by SOLVE but it is used by the subroutines that build and solve the system of linear equations in one call like LINSOL (8.2).
So this value refers to the CG-type solver for the spectral element method with FEM preconditioner. In fact this is an under-subroutine of PCGRAD (8.7)
Possible values:

0 Only error messages are printed

1 A little amount of information of the iteration method is printed

2 A maximal amount of information of the iteration method is printed

Pos. 38: MAX_ITER_SPECTRAL maximum number of iterations for spectral loop as described in Pos. 37.

Pos. 39: isingular
Possible values:

0 standard

1 replace zero pivots by machine accuracy

Pos. 40: Sequence number of intmat in case of ILU preconditioning with different structure than that of the matrix
If NUMSUBEQ $>$ 0:

The next 40 × NUMSUBEQ positions contain information about each subequation separately.

Pos. IPOSPAIRS to IPOSPAIRS+2×NPAIRS-1: Array IPAIRS of size (2,NPAIRS) contains the pairs of nodal points that are connected. The lowest point with the threshold pressure first the other one last.
Hence INPSOL(IPOSPAIRS), INPSOL(IPOSPAIRS+1) contains the first pair of nodes or user points from below to the top, followed by the second pair and so on.

RINSOL Array containing the real user input if iread=-1. The storage of rinsol depends on inpsol.
The following positions are used.

1. eps Required accuracy for the iteration process
2. droptol Drop tolerance with respect to Y12M or in the case of an iteration method the minimum value of the diagonal of the preconditioning matrix.
   In case of iteration_method: adapt diagonal.
   In case of al_gcr: gamma.
3. alpha Parameter for overrelaxation. If zero the value stored in common CITER is used.
   In case of modified ILU this defines the factor f.
   In case of al_gcr: μ.
4. beta Parameter for overrelaxation. If zero the value stored in CITER is used.
   in case of cg: minimum eigenvalue to be used in termination criterion.
5. omega Parameter for overrelaxation. If zero the value stored in CITER is used.
   in case of cg: Accuracy with respect to the computation of the projection vectors
6. alam Parameter for overrelaxation. If zero the value stored in CITER is used.
   in case of cg: Threshold value. All elements in the projection vector with absolute value less than this value are ignored
7. amin Parameter for overrelaxation. Indicates the minimum value for the constraint. See INPSOL(18)
   in case of cg: Value of EPS2.
8. amax Parameter for overrelaxation. Indicates the maximum value for the constraint. See INPSOL(19)
   In case of a spectral element method solved by a CG-type equation with FEM preconditioner, this position contains the accuracy of global CG loop.
   In case of shifted laplace: Real part of shift
9. eps_corr Required accuracy for the defect correction process. Is only used if iinsol(21)>0
   In case of shifted laplace: Imaginary part of shift
10. eps_corr_sol Required accuracy for the iteration method in the defect correction steps (Not the first one). Is only used if iinsol(20)>1 and method≠0

RPOSPAIRS ... RPOSPAIRS-1+NPAIRS: Array threshold contains the threshold values corresponding to the pairs of special boundary conditions for the over-pressures.

MATR Standard SEPRAN array, containing information of the large matrix.
If defect correction is applied, matr must contain two matrices and hence must be declared as matr(5,2).
matr(.,1) corresponds to the matrix to be inverted, for example the upwind matrix.
matr(.,2) corresponds to the matrix to be used for the correction, for example the central matrix.

ISOL Standard SEPRAN array containing information of the solution array. At output the solution array has been filled.

IRHSD Standard SEPRAN array, containing information of the large vector.
If defect correction is applied, irhsd must contain two vectors and hence must be declared as irhsd(5,2) irhsd(.,1) corresponds to matr(.,1) including the effect of boundary conditions.
irhsd(.,2) corresponds to matr(.,2) including the effect of boundary conditions.

INTMAT Array containing information of the structure of the large matrix. Output of a SEPRAN starting subroutine (Section 4.2) or subroutine COMMAT (4.5).
In case of a direct method (SOLUTION_METHOD=0) the matrix must be stored as profile matrix (METHOD = 1, 2, 3 or 4), in case of an iterative method of CG type (SOLUTION_METHOD=1,2,3,4) the matrix must be stored as a compact matrix (METHOD = 5, 6, 7 or 8).

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

IREAD With this parameter it is indicated if array INPSOL is used or that input is read from the standard INPUT file.
Possible values:

0	The input is read from the standard input file
>0	The input as read from the standard input file by subroutine SEPSTN (4.2.3) is used. The value of iread defines the sequence number.
-1	No input is read, all information from array INPSOL is used.
-2	The input is read from the standard input file This input is stored in arrays INPSOL and RINSOL Position 1 of both arrays must contain the declared length.

The input read from the standard input file, is the input corresponding to the main keyword SOLVE. It is described in Section 3.2.8 of the Users Manual.

Input

The arrays INPSOL, RINSOL, MATR, IRHSD, INTMAT, KMESH, KPROB must have been filled.
IREAD must have a value.

Output

The solution vector corresponding to ISOL has been filled.

8.4 Direct linear solvers (subroutine SOLVE)

Description

A system of linear equations will be solved using a direct method (Gaussian elimination). The L-U decomposition of the matrix will be stored.

Call

```
CALL SOLVE ( IPOSST, MATR, ISOL, IRHSD, INTMAT, KPROB )
```

INTEGER IPOSST, MATR(5), ISOL(5), IRHSD(5), INTMAT(5), KPROB(∗)

IPOSST Choice parameter consisting of two parts IPOS and ITRANS according to:
IPOSST = IPOS + 10 × ITRANS, with:

IPOS When the matrix is symmetric (storage scheme METHOD = 1) the matrix may be positive definite. In that case IPOS = 1 means matrix positive definite, a Cholesky decomposition is computed, otherwise IPOS = 0 a $L-D-L^{th}$ decomposition is computed that costs more computing time.

ITRANS Indication how local transformations must be carried out.
Possible values:

0 Standard situation, the solution array is transformed back to the "original" degrees of freedom.

1 The back transformation is skipped. This may be necessary for some iteration procedures.

2 The solution process is skipped, only the backtransformation is carried out.

MATR Standard SEPRAN array, containing information of the large matrix.

ISOL Standard SEPRAN array containing information of the solution array. At output the solution array has been filled.

IRHSD Standard SEPRAN array, containing information of the large vector.

INTMAT Array containing information of the structure of the large matrix. Output of a SEPRAN starting subroutine (Section 4.2) or subroutine COMMAT (4.5).

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

Input

The arrays MATR, IRHSD, INTMAT, and KPROB must have been filled.
IPOSST must have a value.

Output

The solution vector corresponding to ISOL has been filled.

8.5 Iterative linear solvers of CG-type (subroutine CONGRD)

Description

A system of linear equations will be solved by iterative conjugate gradient-type methods. This subroutine can also be used for non-symmetric matrices, in which case a non-symmetric variant of conjugate gradients (Bi-CGSTAB) is used or alternatively the CGS or the GMRES method. The possibility to increase the speed of convergence by preconditioning is available. It is supposed that the matrix is stored as a compact matrix, i.e. METHOD = 5, 6, 7 or 8.

WARNING: the method of conjugate gradients can only converge when all eigenvalues of the matrix to be solved are in the positive half-plane.
CONGRD is an old SEPRAN subroutine. Subroutine SOLVEL (8.3) performs all the tasks of CONGRD.

Call

```
CALL CONGRD ( ICHOICE, ICONTR, MATR, ISOL, IRHSD, INTMAT, KPROB, EPS, NMAX,
              IRESTR )
```

INTEGER ICHOICE, ICONTR(10), MATR(5), ISOL(5), IRHSD(5), INTMAT(5), KPROB(∗), NMAX, IRESTR

DOUBLE PRECISION EPS

ICHOICE Choice parameter. Possibilities:

- 0 Standard situation, the standard conjugate process (symmetrical or not) is used without preconditioning.
- 1 The standard conjugate process (symmetrical or not) is used with a standard preconditioning. The preconditioning matrix is computed from the matrix.
- 2 The standard conjugate process (symmetrical or not) is used with a standard preconditioning. The preconditioning matrix is stored as permanent array and its reference is put in MATR(4). It may be reused in a next call with ICHOICE = 3.
- 3 The standard conjugate process (symmetrical or not) is used with the preconditioning that is already stored in MATR. This is only the case if CONGRD has been called before with ICHOICE = 2, or ICHOICE < 0 and ICONTR(10) = 1.
- -1 Non-standard situation. Information of the process must be stored by the user in array ICONTR.
- -2 Non-standard situation. Information of the process must be stored by the user in array ICONTR. Extra choice: the relative error definition.

ICONTR This array of length 10 or 12 is only used when ICHOICE = -1 or -2.
The user must fill the positions 1 to 10 with the following information:

Pos. 1 Number of iterations that have been executed before by CONGRD.
When no iterations have been performed, ICONTR(1) must be equal to 0.
When ICONTR(1)=-1 the subroutine supposes that no iterations have been executed, but stores information in array IRESTR for a possible restart. Also in the case ICONTR(1)>0 information is stored in array IRESTR.
At this moment only ICONTR(1)=0 is available.

Pos. 2 ISTART: Control parameter concerning the start of the conjugate gradient process. ISTART is only used if ICONTR(1)=0. Possibilities:

- 0 The process starts with a starting vector **0** for all internal points.
- 1 The process starts with an explicitly given start vector.

Pos. 3 Indication of whether the process must stop if an essential error occurs during the iteration process (0), or whether control is resubmitted to the main program (1). In the latter case information concerning the error is stored in ICONTR(6).

Pos. 4 IPRINT: Control parameter concerning the amount of output that must be printed. Possibilities:

- <0 No print output.
- 0 Only fatal errors will be printed.
- 1 Minimal extra output, giving some information concerning the iteration process.
- 2 Gives a maximal amount of output concerning the iteration process.

Pos. 5 ICONTR(5) consists of two parts: IPREC and METHOD according to ICONTR(5) = IPREC + 10 × METHOD, with
IPREC: Control parameter concerning the type of preconditioning to be used. Possibilities:

- 0 No preconditioning.
- 1 The matrix is pre-multiplied by a diagonal matrix such that all diagonal elements become equal to 1. The corresponding diagonal matrix is used as preconditioning matrix.
- 2 As preconditioning matrix is an incomplete LU decomposition used, where only the diagonal is changed. (Efficient version of Eisenstat).
- 3 As preconditioning matrix is an incomplete LU decomposition used.
- 4 As preconditioning matrix is the matrix resulting from the Gauss-Seidel iteration process used.
- 5 Incomplete LU-decomposition using a pre-assembled decomposition according to a new method of Rik Kaasschieter.
- 6 A standard (=default) preconditioner.
- 7 See IPREC=2, however, in this case a modified incomplete LU decomposition is used.

METHOD: Control parameter concerning the type of iteration method used.
Possibilities:

- 0 Standard method, i.e. conjugate gradients for symmetric problems and Bi-CGSTAB for non-linear problems.
- 1 If the storage method for non-symmetric matrices is 6, the CGS method is used instead of Bi-CGSTAB.
- 2 If the storage method for non-symmetric matrices is 6, the GMRES method is used instead of Bi-CGSTAB. In that case ICONTR(9) must also be filled if ICHOICE < 0.
- 3 If the storage method for non-symmetric matrices is 6, the GMRESR method is used instead of Bi-CGSTAB. The GMRESR method is an efficient variant of the standard GMRES method in which the inner loop is formed by a GMRES method, and the outer loop is formed by a kind of polynomial preconditioning also based on the GMRES method. This combination might decrease the number of vectors and hence the space needed for the GMRES process. In that case ICONTR(9), ICONTR(11) and ICONTR(12) must also be filled if ICHOICE < 0.

If ICHOICE=-2 then the 7^{th} and 8^{th} position must have been filled as well, viz:

Pos. 7 Control parameter concerning the relative error in the termination criterion (IRELER). Possibilities:

- 0 relative error is taken with respect to $\| x \|$.
- 1 The relative error is taken with respect to $\| res_0 \|$, where res_0 is the initial residual.
- -1 The relative error is taken with respect to $\| b \|$.

Pos. 8 MATSYM Is used only when the non-symmetric storage scheme JMETOD=9,10,11,12 is used.

- 0 The matrix is supposed to be non-symmetric.
- 1 The matrix is symmetric.

Pos. 9 MGMRES Is used only when the GMRES method is used. It indicates the maximum number of vectors that is used in the orthogonalization process. If this number is too small the method does not converge or converges too slow. If this number is too large a large amount of storage is used, which may decrease the performance or even result in an error message because of lack of space in the buffer array.
MGMRES must have a value between 2 and 200.

Pos. 10 KEEP With this parameter the user may indicate whether the preconditioning matrix must be destroyed, kept or that an old preconditioning matrix must be used. Possibilities:

- 0 The preconditioning matrix is destroyed.
- 1 The preconditioning matrix is kept. A reference is written in MATR(4).
- 2 The preconditioning matrix as computed before is used. It is supposed that the reference for this matrix has been stored in MATR(4). This may have been done either by a preceding call with ICHOICE<0 and ICONTR(10)=1, or a call with ICHOICE=2.

Pos. 11 NTRUNC Is only used if the GMRESR method is applied. It indicates the maximum number of vectors to be used in the outer orthogonalization process. If this maximum has been reached each new vector replaces a suitable old one. MGMRES indicates in this case the number of outer iterations at which the process is restarted completely. As a consequence MGMRES should be much larger than NTRUNC. Typical values for NTRUNC and MGMRES are respectively 20 and 200.

Pos. 12 NINNER Is only used if the GMRESR method is applied. It indicates the maximum number of inner iterations of the GMRES method. In fact this parameter defines the inner Krylov space and hence is an important parameter with respect to work space needed. A typical value for NINNER is 5.

If ICONTR is used, at output the following information is stored in the array.

Pos. 1 Number of iterations executed.

Pos. 6 Error indication. Possibilities:

- 0 No fatal errors.
- 1 The desired preconditioning matrix does not exist. It is advisable to use another preconditioning.
- 2 The conjugate gradient process has terminated because the matrix is not positive definite (symmetrical case), or because there are eigenvalues with negative real part.
- 3 The required accuracy has not been reached within NMAX steps.

MATR Standard SEPRAN array, containing information of the large matrix.

ISOL Standard SEPRAN array containing information of the solution array.
If ICONTR(2)=1 also the start vector must have been filled in ISOL.
At output the solution array has been filled with the last iteration computed.

IRHSD Standard SEPRAN array, containing information of the large vector.

INTMAT Array containing information of the structure of the large matrix. This matrix must have a compact structure, METHOD = 5-8.
Output of a SEPRAN starting subroutine (Section 4.2) or subroutine COMMAT (4.5.2) (4.5).

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

EPS The absolute value of EPS gives the required accuracy. If EPS > 0, the accuracy is based on the absolute error, if EPS < 0 the accuracy is based on the relative error.

NMAX Maximal number of iterations that may be performed by the conjugate gradient process. In any way no more than n (is number of free degrees of freedom) iterations are carried out.

IRESTR Not yet used.

Input

The arrays MATR, IRHSD, INTMAT and KPROB must be filled.
When ICHOICE=-1, array ICONTR must have been filled by the user.
Boundary conditions must have been filled in ISOL, and depending on the value of ICHOICE and ICONTR also a start vector must have been filled in array ISOL.
ICHOICE, EPS and NMAX must have a value.

Output

Information concerning the solution is stored in array ISOL.
If ICHOICE=-1 some information has been stored in array ICONTR.

Warning

The conjugate gradient method converges only when all eigenvalues of the matrix are in the positive half plane. The method is an iterative method that may converge, but also the convergence may fail. In that case the user should use either another preconditioning, or another iterative method or if no other procedure is available, a direct method as provided by subroutine SOLVE.
In general the conjugate gradient method is a fast converging method for positive definite problems. One must, however, keep in mind that the method is only faster than a direct method for large 2D problems (more than 1000 degrees of freedom) or moderate 3D problems (from 300 degrees of freedom). However, when a good starting vector is available as for example in time-dependent problems or non-linear iteration processes the method may gain in time for even smaller problems. In any way iterative methods require far less memory than direct methods.

In general the use of a preconditioning matrix accelerates the speed of convergence considerably. Which of the preconditionings must be used can not be said a-priori. As a default the incomplete modified LU decomposition with only the diagonal changed (Eisenstat version) is used, since this method proved to be faster in a number of test cases. For a specific example the user may compare the time needed for CONGRD for a number of preconditionings and then decide which preconditioning fittest him most.

8.6 Overrelaxation based solvers for optimization with constraints

In this chapter we treat two old-fashioned SEPRAN subroutines to solve linear systems with or without constraints by the overrelaxation method. At present calls to these subroutines may be replaced by a call to SOLVEL (8.3).
It concerns the following subroutines:

OVERRL (8.6.1) Solution of a system of linear equations by overrelaxation.

OVERCS (8.6.2) Solution of a system of linear equations with constraints by overrelaxation. This subroutine is meant for optimization with constraints.

8.6.1 Subroutine OVERRL

Description

A system of linear equations will be solved by the iterative overrelaxation method. An approximation of the optimal overrelaxation factor ω is computed.
warning: generally the overrelaxation methods converges only when all eigenvalues of the matrix to be solved are in the positive half-plane.

Call

```
CALL OVERRL( ICHOICE, MATR, IRHSD, ISOL, INTMAT, KMESH, KPROB, EPS, NMAX )
```

INTEGER ICHOICE, MATR(5), ISOL(5), IRHSD(5), INTMAT(5), KPROB(∗), NMAX, KMESH(∗)

DOUBLE PRECISION EPS

ICHOICE Choice parameter consisting of two parts ISTART and IPRINT according to:
ICHOICE = ISTART + 10 × IPRINT with

ISTART Defines how the start vector must be created. Possible values:

0 The subroutine is self-starting. As initial guess the zero-vector is used.

1 The user is supposed to give the initial guess himself in the solution vector.

IPRINT Defines the amount of output. Possible values:

0 Only error messages and warnings are given.

1 Information about the iteration process is printed.

MATR Standard SEPRAN array, containing information of the large matrix.

ISOL Standard SEPRAN array containing information of the solution array.
If ICHOICE = 1 also the start vector must have been filled in ISOL.
At output the solution array has been filled with the last iteration computed.

IRHSD Standard SEPRAN array, containing information of the large vector.

INTMAT Array containing information of the structure of the large matrix. This matrix must have a compact structure, METHOD = 9 or 10.
Output of a SEPRAN starting subroutine (Section 4.2) or subroutine COMMAT (4.5.2).

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

KMESH Output of the mesh generator.

EPS The absolute value of EPS (ϵ) defines the accuracy required. If EPS < 0, the relative error is considered, otherwise the absolute error.
During the process the in absolute value largest eigenvalue λ_{MAX} is computed. The process is stopped when: $||u^{k+1} - u^k|| < (1 - \lambda_{MAX})\epsilon$ (absolute criterion), or $\frac{||u^{k+1} - u^k||}{||u^{k+1}||} < (1 - \lambda_{MAX})\epsilon$ (relative criterion).
or when the number of iterations exceeds NMAX.
The norm that is used is the max norm.

NMAX Maximal number of iterations that may be performed by the overrelaxation process. When the number of iterations equals NMAX+1 the process is stopped, the estimation of λ_{MAX} is printed as well as the difference: $||u^{k+1} - u^k||$ (EPS>0) or $\frac{||u^{k+1} - u^k||}{||u^{k+1}||}$ (EPS<0).

Besides the parameters in the parameter list, the user may utilise parameters in the common block CITER, in order to control the iteration process according to his own wishes.

```
COMMON  /CITER/ OMEGA, ALPHA, BETA, ALAM, NITER, NITER1, NITER2
```

OMEGA overrelaxation parameter ω used. The starting subroutines give ω the value 1. OMEGA is changed during the process, unless NITER1 = -2.

ALPHA The factor α, see method.

BETA The factor β, see method.

ALAM The approximation of the largest eigenvalue, see method.

NITER Number of iterations performed.

NITER1 The factor N_1, see method.

NITER2 The factor N_2, see method.

Input

The arrays MATR, IRHSD, INTMAT, KMESH and KPROB must be filled.
Boundary conditions must have been filled in ISOL, and depending on the value of ICHOICE also a start vector must have been filled in array ISOL.
ICHOICE, EPS and NMAX must have a value.

Output

Information concerning the solution is stored in array ISOL.

Method

For a description of the method used to compute the overrelaxation parameter, we refer to the Users Manual, Section 3.2.8.

8.6.2 Subroutine OVERCS

Description

A system of linear equations with linear constraints will be solved by an iterative projection over-relaxation method. An approximation of the optimal overrelaxation factor *omega* is computed.

warning: generally the overrelaxation methods converges only when all eigenvalues of the matrix to be solved are in the positive half-plane.

Call

```
CALL OVERCS( ICHOICE, MATR, IRHSD, ISOL, INTMAT, KMESH, KPROB, EPS, NMAX,
             IWORK, WORK )
```

INTEGER ICHOICE, MATR(5), ISOL(5), IRHSD(5), INTMAT(5), KPROB(∗), NMAX, KMESH(∗), IWORK(4)

DOUBLE PRECISION EPS, WORK(4)

ICHOICE, MATR, IRHSD, ISOL, INTMAT, KMESH, KPROB, NMAX See OVERRL (8.6.1).

EPS See OVERRL (8.6.1).

IWORK,WORK Input arrays of length 4 respectively 2 to be filled by the user. In these arrays information concerning the constraints must be stored by the user. IWORK and WORK must be filled as follows:

IWORK(1): Degree of freedom that must be limited. If IWORK(1)=0, the constraint is applied to all degrees of freedom, otherwise (IWORK(1)>0) it is applied to the $IWORK(1)^{th}$ degree of freedom in each nodal point.

IWORK(2): Indication of the type of constraint. Possibilities:

- 0 No constraint.
- 1 The solution **u** satisfies: $\mathbf{u}(\mathbf{x}) \geq \mathbf{a}(\mathbf{x})$
- 2 The solution **u** satisfies: $\mathbf{u}(\mathbf{x}) \leq \mathbf{b}(\mathbf{x})$
- 3 The solution **u** satisfies: $\mathbf{a}(\mathbf{x}) \leq \mathbf{u}(\mathbf{x}) \leq \mathbf{b}(\mathbf{x})$

IWORK(3): Information of the lower bound $\mathbf{a}(\mathbf{x})$. Possibilities:

- 0 $\mathbf{a}(\mathbf{x}) = \mathbf{0}$
- >0 $\mathbf{a}(\mathbf{x})$ is equal to a constant. The value of the constant must be stored by the user in WORK(1)
- <0 $\mathbf{a}(\mathbf{x})$ is a function of the position. The user must provide a function subroutine FUNC according to the rules given in the INTRODUCTION Section 5.5.4 in order to compute the function $\mathbf{a}(\mathbf{x})$. The parameter ICHOICE in FUNC is equal to -IWORK(2).

IWORK(4): Information of the upper bound $\mathbf{b}(\mathbf{x})$. Possibilities:

- 0 $\mathbf{b}(\mathbf{x}) = \mathbf{0}$
- >0 $\mathbf{b}(\mathbf{x})$ is equal to a constant. The value of the constant must be stored by the user in WORK(2)
- <0 $\mathbf{b}(\mathbf{x})$ is a function of the position. The user must provide a function subroutine FUNC according to the rules given in the INTRODUCTION Section 5.5.4 in order to compute the function $\mathbf{b}(\mathbf{x})$. The parameter ICHOICE in FUNC is equal to -IWORK(3).

With respect to common CITER the reader is referred to OVERRL (8.6.1).

Input

The arrays MATR, IRHSD, INTMAT, KMESH, IWORK, WORK and KPROB must be filled.
Boundary conditions must have been filled in ISOL, and depending on the value of ICHOICE also a start vector must have been filled in array ISOL.
ICHOICE, EPS and NMAX must have a value.

Output

Information concerning the solution is stored in array ISOL.

Method

For a description of the method used to compute the overrelaxation parameter, we refer to the Users Manual, Section 3.2.8.

8.7 Solution of spectral element system by an iterative solver (subroutine PCGRAD)

Description

Subroutine PCGRAD may be used to solve a spectral element discretized system. PCGRAD solves the system using a conjugate gradient method. In general the preconditioner is the finite element stiffness matrix build on the same Legendre-Gauss-Lobatto integration points as the spectral system.

Heading

```
subroutine pcgrad ( iu0, iun, matfe, kmesh, kprob, intmat, irhsd, iuser, user, eps )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), IUSER(∗), INTMAT(∗), IRHSD(∗), IU0(∗), IUN(∗), MATFE(∗)

DOUBLE PRECISION EPS, USER(∗)

IU0 Contains information of the initial approximation.

IUN Contains information of the solution.

MATFE Contains information of the finite-element matrix.

KMESH Standard SEPRAN array containing information about the spectral element mesh.

KPROB Standard SEPRAN array containing information about the problem definition.

INTMAT Contains information about the structure of the matrix.

IRHSD Contains information about the right-hand-side vector.

IUSER Communication array for the element subroutines.

USER Double precision communication array for the element subroutines.

EPS Convergence criterium for the convergence loop

Input

The arrays IU0, MATFE, KMESH, KPROB, INTMAT, IRHSD, IUSER, USER must have been filled.
The tolerance parameter EPS must have a value.

Essential boundary conditions must have been filled in the array corresponding IU0.
The spectral right-hand-side vector corresponding to IRHSD should be created without boundary conditions.

Output

Array IUN is filled with information concerning the solution.

9 Solution of non-linear equations

9.1 Introduction

In this chapter the various SEPRAN subroutines to solve non-linear systems of equations will be treated. In fact each non-linear problem is solved as a series of building and solving of linear equations. Through iteration the final result must be reached.
Unfortunately, an inherent aspect of non-linear problems is that the iteration process might not converge or even worse, the iteration process may converge to the wrong solution. The user must be completely aware of these properties.

If a non-linear problem does not converge there are several ways to try to get a convergent process. The main possibilities are:

continuation If a non-linear process does not converge it is a good method to find a relevant parameter for the process which influences the convergence. For example the Reynolds number in a laminar flow is of extreme importance for the convergence. A low Reynolds number does not produce any problems with respect to convergence, whereas a high Reynolds number does. The continuation method implies that one starts with a value of the relevant parameter for which the iteration process converges (for example with a low Reynolds number). During the iteration process the relevant parameter is smoothly adapted (in case of the Reynolds number increased) until the actual parameter is reached. In this way the iteration process may converge.

time-dependence Another option is to make the stationary non-linear equation time-dependent by adding a first order time-derivative. Since time-dependent methods converge more smoothly, especially if the time steps are relatively low, this might be a good alternative. Still time-dependent methods usually take more time and if a direct non-linear iteration is possible such a method is recommended.

In this chapter we deal with the following paragraphs:

9.2 General subroutines for the solution of non-linear equations

In this section we consider the SEPRAN subroutines that are available for the solution of non-linear problems. Besides the old-fashioned SEPRAN subroutine NONLIN, treated in Section 9.3, it concerns the following subroutines:

NLNSOL (9.2.1) This subroutine is the simplest subroutine for the solution of non-linear problems. Each time it is called it reads input from the standard input file.

NLNPRB (9.2.2) is the natural extension of NLNSOL. It is also able to read the input from the standard input file, but besides that input read before by the SEPRAN starting subroutines SEPSTx are also recognized.

9.2.1 Subroutine NLNSOL

Description

Solves a non-linear problem by iteration. In each step of the process a new matrix and right-hand side is built, and a system of linear equations is solved. Convergence of the iteration process is controlled according to the users wishes.
All input is read from the standard input file each time the subroutine is called.

Heading

```
subroutine nlnsol ( kmesh, kprob, intmat, isol )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), ISOL(5), INTMAT(5)

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

INTMAT Array containing information of the structure of the large matrix. Output of a SEPRAN starting subroutine (Section 4.2) or subroutine COMMAT (4.5.2).

ISOL Standard SEPRAN array containing information of the solution vector. In general isol must have been filled with the essential boundary conditions before the call of nlnsol. At output the solution vector corresponding to isol has been filled.

Input

Subroutine NLNSOL reads input from the standard input file each time the subroutine is called. The description of the input can be found in the Users Manual Section 3.2.9 corresponding to the keyword NONLINEAR_EQUATIONS.
Mark that although this subroutine itself expects that the input is given after the keyword
END_OF_SEPRAN_INPUT still input about linear solver and coefficients must have been read in the first part. The reason is that the input for NLNSOL refers to sequence numbers corresponding to these input parts.
The arrays KMESH, KPROB and INTMAT must have been filled.
Essential boundary conditions must have been filled into array ISOL.

Output

The computed solution has been stored in the array corresponding to ISOL.

9.2.2 Subroutine NLNPRB

Description

Solves a non-linear problem by iteration. In each step of the process a new matrix and right-hand side is built, and a system of linear equations is solved. Convergence of the iteration process is controlled according to the users wishes.
The user may indicate whether the input is read from the standard input file in each call or that the input is read by the SEPRAN initialization subroutines SEPSTx.

Heading

```
subroutine nlnprb ( kmesh, kprob, intmat, isol, iinsol )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), ISOL(5,∗), INTMAT(5), IINSOL(∗)

KMESH Standard SEPRAN array in which information of the mesh is stored. KMESH must have been filled before either by a call to subroutine MESH or by a mesh reading subroutine.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

INTMAT Array containing information of the structure of the large matrix. Output of a SEPRAN starting subroutine (Section 4.2) or subroutine COMMAT (4.5.2).

ISOL Standard SEPRAN array containing information of the solution vector or a series of solution and other vectors. In general the vector in isol that is used to store the solution must have been filled with the essential boundary conditions before the call of nlnprb. At output the solution vector corresponding to isol has been filled. ISOL is a more-dimensional array. Each array (.,i) refers to another array with sequence number i.

IINSOL Input array defined by the user.
With this array the user may indicate which actions he wants to be performed by this subroutine.
IINSOL must be filled as follows:

Pos. 1: Last entry number that has been filled by the user. If 0 or 1 only defaults are used

Pos. 2: IREAD Defines how the input must be read.
Possible values:

0: All SEPRAN input has been read by subroutine SEPSTN (4.2.3) until END_OF_SEPRAN_INPUT or end of file has been found. This input is used.
Instead of reading the input by subroutine SEPSTN, the user may also fill the arrays IINPUT and RINPUT himself and store them in the arrays IBUFFR and BUFFER by subroutine FILLINP 4.7. The description of the contents of IINPUT and RINPUT is given at the end of this section.

1: The input is read as described in the Users Manual Section3.2.9.

Default value: 0

Pos. 3: ISEQNR, This parameter is only used if IREAD=0. It indicates the sequence number of the input for the non-linear solver to be used.
Default value: 1

Pos. 4: IPROB, Problem sequence number.
Default value: 1

Pos. 5: IVECTR, sequence number of solution vector with respect to array ISOL.
Default value: iprob.

Pos. 6: MAXVEC, Maximum number of vectors stored in isol
Default value: iprob

Pos. 7: IVEC_REACT, indicates if reaction forces must be computed (> 0) or not (0).
If IVEC_REACT > 0, IVEC_REACT defines the sequence number of the vector in ISOL in which the reaction forces must be stored.
Default value: 0

Input

Depending on the value of IREAD subroutine NLNPRB may read input from the standard input file when the subroutine is called. The description of the input can be found in the Users Manual Section 3.2.9 corresponding to the keyword NONLINEAR_EQUATIONS.
The arrays KMESH, KPROB and INTMAT must have been filled.
Essential boundary conditions must have been filled into array ISOL as well as all vectors that are used to compute the coefficients for the equations.

Output

The computed solution has been stored in the array corresponding to ISOL.

The arrays IINPUT and RINPUT

Usually the arrays IINPUT and RINPUT are filled by reading of the standard input file. However, the user may fill these arrays himself and put them into the buffer arrays IBUFFR and BUFFER through the call of FILLINP 4.7. The structure of these arrays is as follows:

IINPUT Global contents:

Pos. 1-25 General information for all problems

Pos. 26-25+NCOPEQ Starting addresses in IINPUT of the specific information for each of the coupled equations. If the starting address is 0 no special information is expected.

Pos. 26+NCOPEQ... Specific information of the coupled equations.

Detailed contents.
Pos. 1-25 General information for all problems

1 NCOPEQ Number of coupled equations to be solved by this subroutine.
For each iteration all equations are solved in the natural sequence, where the output of the preceding solutions may be used as input for the new ones.

2 MAXITER Maximum number of iterations allowed.

3 IPRINT Defines the amount of output produced. Possible values:

-1 All output information is suppressed, even the message of no-convergence

0 No information about the iteration process is printed except in the case of an error.

1 The number of iterations performed is printed

2 Some extra information about the iteration process is printed for each iteration

3 See 2, in this case also the ratio of two succeeding iterations is printed

100 Contains the maximum of preceding values and also prints the solution in each iteration

4 IRETURN Indicates if the subroutine must stop if no convergence has been found (0) or that control must be given back to the main program (1)

5 MINITER Minimum number of iterations allowed

6 ICRITER Type of criterion to be used for the termination. ICRITER consists of two parts ICRIT and IRESCRIT according to: ICRIT + 100*IRESCRIT
Possible values for ICRIT

0 Absolute difference between two succeeding iterations

1 Relative difference between two succeeding iterations

2 Absolute value of right-hand side

3 Absolute value of right-hand side divided by solution

4 Absolute value of energy increment (u,f)

5 Absolute value of energy increment (u,f) divided by solution

IRESCRIT Extra type of criterion based on residual. To reach convergence both the standard criterion as well as the extra criterion must be satisfied.
Possible values:

0 No residual criterion

1 Absolute residual criterion

2 Relative residual criterion, i.e. based on $\frac{||residual||}{||solution||}$

The corresponding parameter eps is stored in rinsol(3)

7 ISEQSOL Sequence number of linear solver to be used

8 IVECTR Sequence number of first vector Unless stated otherwise the vector numbers are given by IVECTR, IVECTR+1, ... , IVECTR+NCOPEQ-1

9 CALLUSR Indicates if the user subroutine funnln must be called in each step (1) or not (0)

10 MAXVEC Maximum number of vectors stored in ISOL

11 ISEQOUT Sequence number of intermediate output to sepcomp.out
If 0, no output is written

12 IPLOT Indicates if the solution must be plotted during the iterations in order to trace the iteration process.
Possible values:

0 No plots are made

1 If the solution is a scalar a contour plot is made. If the solution is a vector a vector plot of the solution is made.
Otherwise no plot is made.

IPLOT>0 has only been implemented for NDIM=2

13 METHOD_NL Indicates type of iteration method to be applied.
Possible values:

0 Standard method $A^k x^{k+1} = f^k$

1 Newton method: dx = 0 at boundary.
$A^k dx = f^k$ $x^{k+1} x^k + dx$

2 Incremental Newton method: dx^1 contains the boundary condition for the increment.
$A^k dx^k = f^k$
$xtotal^{k+1} = xtotal^k + dx^k$
$dx^k = 0$ at boundary
In this case we need 2 vectors:
ivectr: containing dx
ivec_tot: containing the total solution

14 IVEC_TOT Sequence number of total solution in ISOL in case of METHOD_NL = 2

15 IUSEUSR This parameter indicates if the arrays IUSER and USER from the parameter list are used (1) or if the parameter ISEQFILCOF is used (0).

16 ICOPYOLD If icopyold ¿ 0, the solution in the last iteration before the final iteration is stored in array ISOL.
The sequence numbers ICOPYOLD ...ICOPYOLD+NCOPEQ-1 in ISOL are used to store the NCOPEQ solution vectors.
The user must be carefull to create ISOL with sufficient length. No check is possible

17 LIMIT_SOLUTION Limit the solution to minimum and maximum values.
Posible values:

0 no limiting

1 only the end result

18 ISUBDIVIDE indicates if quadratic elements must be treated as a conglomerate of linear elements (1) or as quadratic elements itself (0).

19-24 Not yet defined

25 NITER Output parameter: indicates the number of iterations performed

Pos. 26+NCOPEQ + ... specific information about the various equations.
For each of the equations the following information is required:

1 ICRITER Local type of criterion.
(if -1 the global criterion is used)

2 IPOSEPS Position of local epsilon in array RINSOL
(if 0 the global eps is used)

3 IPOSOMEG Position of local omega in array RINSOL
(if 0 the global omega is used)

4 ISEQSOLVE Sequence number of linear solver call to be used.
If 0 the global one is used

5 ISEQFILCOF Sequence number of FILCOF call to be used in the first iteration.
If 0, FILCOF is not called

6 IVECTR Sequence number of vector to be solved.
If 0, the corresponding global value is used

7 NUMCHANCOF Maximum number of calls of CHANCOF during the iterations.
If 0, change coeff is not called

8 ISTARTCHAN Starting address of information about the calls to CHANCOF minus one

9 NUMPRESBC Maximum number of calls to PRESBC during the iterations.
If 0, PRESBC is not called

10 ISTARTPRES Starting address of information about the calls to PRESBCF minus one

11 METHOD_NL See global value
if 0 the global value is used

12 IVECTOT See global value
if 0 the global value is used

13 IPOSRELEPS Position of local residual epsilon in array RINSOL
(if 0 the global eps is used)

14 LIMIT_SOLUTION Limit the solution to minimum and maximum values.
Posible values:

0 no limiting

1 only the end result

15 ISUBDIVIDE indicates if quadratic elements must be treated as a conglomerate of linear elements (1) or as quadratic elements itself (0).

16-25 Not yet used.

ISTARTCHAN +2*j-1 iteration number for call j

ISTARTCHAN +2*j sequence number for call j

ISTARTPRES +2*j-1 iteration number for call j

ISTARTPRES +2*j sequence number for call j

RINPUT At this moment only 3 positions are used.

1 Global accuracy eps

2 Global relaxation parameter omega

3 Accuracy with respect to residual

9.3 Old fashioned subroutines to solve non-linear problems (NONLIN)

Description

Subroutine NONLIN can be used for the solution of non-linear stationary problems.
In the case of a linear problem it can be used as a substitute for the subroutines BUILD (7.2) and SOLVE (See Users Manual).
When the problem is non-linear, an iterative procedure is carried out. The subroutines BUILD (7.2) and SOLVE are called during this process.

Call

```
CALL NONLIN ( ITERTP, MAXITR, EPS, IPRINT, IWORK, WORK, INTMAT, ICHSYS,
              IUSER, USER, KMESH, KPROB, ISOL, IELHLP )
```

INTEGER ITERTP, MAXITR, IPRINT, IWORK(∗), INTMAT(5), ICHSYS, IUSER(∗), KMESH(∗), KPROB(∗), ISOL(5), IELHLP(∗))

DOUBLE PRECISION EPS, WORK(∗),USER(∗),

ITERTP Type of iteration procedure. ITERTP consists of four parts JTERTP, IRELAT, IRESID and ISEPAR according to:
ITERTP = JTERTP + IRELAT × 10 + IRESID × 100 + ISEPAR × 1000, with the following meaning:

JTERTP Indicates if the problem is treated as a linear problem or non-linear problem. Possible values:

0 The problem is treated as a linear problem. Only one iteration is carried out. The subroutines BUILD (7.2) and SOLVE are called only once.

1 Standard iterative procedure with at most MAXITR iterations. During each iteration BUILD (7.2) and SOLVE are called. A relaxation parameter to control the iteration process may be used via subroutine FILNLN.
See Section 9.4.

IRELAT Indicates the type or error criterion to be used. Possible values:

0 Absolute error criterion of difference max $|\mathbf{u}_i - \mathbf{u}_{i-1}| < \varepsilon$, $(i \geq 1)$ is tested at each iteration for all nodal points.

1 Relative error criterion of difference $\frac{\max |\mathbf{u}_i - \mathbf{u}_{i-1}|}{\max |\mathbf{u}_i|} < \varepsilon$, (i $\geq$ 1) is tested at each iteration for all nodal points.

IRESID Indicates if the residual must be computed and tested. Possible values:

0 The residual is not computed and hence not tested.

1 Before iteration $i \geq 1$ the relative residual $\frac{||\mathbf{A}\mathbf{u}_{i-1} - \mathbf{f}_{i-1}||}{||\mathbf{u}_{i-1}||} < \varepsilon$ is tested, where $\mathbf{A}$ is the iteration matrix and $\mathbf{f}$ the iteration right-hand side. The residual measures how good the discrete equations are satisfied. Mark that the computation of the residuals is one iteration behind.

ISEPAR Indicates if all unknowns in the equations are treated as one part, or that separate values are distinguished. Possible values:

0 For the norm max $|\mathbf{u}_i - \mathbf{u}_{i-1}|$ all degrees of freedom are taken together.

1 The norms max $|\mathbf{u}_i - \mathbf{u}_{i-1}|$ and, if required, max$|\mathbf{u}_i|$ are determined for each degree of freedom separately. The maximum of the norms (absolute or relative) are tested against ε.

The most complete test is provided for ITERTP = 1111.

MAXITR Maximum number of iterations.
When IPRINT < 0, MAXITR is input **and** output parameter. In that case $|MAXITR|$ is equal to the number of of iterations performed at output.
The iteration stops (and sets MAXITR < 0 in the case IPRINT < 0) if:

i The iteration process diverges after at least 5 iterations.

ii The maximum number of iterations is not sufficient to reach the required accuracy EPS between two succeeding solutions of the system of equations.

EPS The absolute or relative accuracy ε required. The iteration stops when the difference between two succeeding iterations is smaller than EPS.
If IPRINT<0 and the process did not converge after MAXITR iterations, then EPS gets the value $\max|\mathbf{u}_i - \mathbf{u}_{i-1}|$ at output for IRELAT $= 0$, or the value $\frac{\max |\mathbf{u}_i - \mathbf{u}_{i-1}|}{\max |\mathbf{u}_i|}$ for IRELAT $=$ 1.

IPRINT Output control parameter. Possibilities:

0 No special output.

1 The total number of iterations is printed.

2 Extra information of each iteration is printed.

3 The ratio of differences with preceding iteration is printed too.

-1,-2,-3 See IPRINT = 1,2,3, however, the program is not stopped, but control is returned to the main program. See also MAXITR, EPS and ISOL.

IWORK Not yet used.

WORK Not yet used.

INTMAT Output of subroutine SEPSTR (4.2.1) or COMMAT (4.5.2).
When METHOD = 1 in the input of SEPSTR (4.2.1) or JMETOD=1 in the call of COMMAT (4.5.2), the matrix is supposed to be symmetrical and positive definite.

ICHSYS Parameter for subroutine BUILD (7.2). ICHSYS must be equal to IINBLD(2) + 10 × IINBLD(3) + 20 × IINBLD(6) + 40 × IINBLD(7) + 100 × IINBLD(8), with IINBLD the input array for subroutine BUILD (7.2) (7.2)
In general ICHSYS should be equal to 1 when ITERTP=0 and equal to -1 when JTERTP>0. When JTERTP>0 it is necessary that ICHSYS<0.

IUSER In this array integer information concerning coefficients etc. must be stored. See the manual Standard Problems.

USER In this array real information concerning coefficients etc. must be stored. See the manual Standard Problems. The arrays IUSER and USER must be dimensioned.

KMESH Output of mesh generator

KPROB Output of subroutine PROBDF (4.4)

ISOL Array containing information of the solution vector. Essential boundary conditions must be filled in the solution vector corresponding to ISOL either before the call of NONLIN or in the user subroutine FILNLN (9.4) When an iteration process requires an initial estimate, then information concerning this estimate must be filled in array ISOL.
When IPRINT<0 and the iteration process has ended without converging, then ISOL refers to the last computed iteration. Length: 5 positions.

IELHLP Help array for subroutine BUILD (7.2), depending on the value of ICHSYS.

Input

The arrays INTMAT, KMESH, KPROB and if necessary IUSER and USER must have been filled.
When there are essential boundary conditions array ISOL must have been filled.
ITERTP, ICHSYS and when ITERTP>0 MAXITR, EPS and IPRINT must have a value.

Output

ISOL contains information concerning the solution of the non-linear problem.

Subroutines called

Subroutine NONLIN calls the user subroutine FILNLN.

Remarks

If the linearization of the equations is a Picard linearization (i.e. successive substitution) and the ratio of differences C = $\frac{\max |\mathbf{u}_i - \mathbf{u}_{i-1}|}{\max |\mathbf{u}_{i-1} - \mathbf{u}_{i-2}|}$ remains close to 1, the difference $\frac{\max |\mathbf{u}_i - \mathbf{u}_{i-1}|}{\max |\mathbf{u}_i|}$ might be small accompanied with slow convergence. in order to ensure that $\mathbf{u}_i$ is close to the final solution a sharper criterion $\varepsilon_1 = (1 - C)\varepsilon$ must be satisfied.
If you use a relaxation factor ω, $0 < \omega < 2$, then $\varepsilon_1 = \frac{(1-C)}{1-\omega(1-C)}\varepsilon$ is a better criterion.
If the linearization of the equations is a Newton linearization and the iteration $\mathbf{u}_i$ is rather close to the solution, the number of significant digits should, approximately, double each further iteration until accuracy inhibits any more progress.

9.4 Subroutine FILNLN

Description

This user-written subroutine is called by subroutine NONLIN (9.3), during each step of the iteration process. With the aid of this subroutine the user may fill parameters and coefficients in the arrays IUSER and USER and in that way influence the iteration process.
It is also possible to specify essential boundary conditions in FILNLN.

Call

```
CALL FILNLN ( NITER, IUSER, USER, KMESH, KPROB, ISOL, RELAX, DIFFER )
```

INTEGER NITER, IUSER(∗), KMESH(∗), KPROB(∗), ISOL(5)

DOUBLE PRECISION USER(∗), RELAX, DIFFER

NITER Number of iterations performed by NONLIN before the call of FILNLN.
Note: FILNLN is called by NONLIN before **each** iteration. When the user wants to fill the boundary conditions in the solution vector in subroutine FILNLN, in general this is only necessary if NITER = 0.

IUSER In this array the user must store the integer information concerning the coefficients and other relevant parameters. See the manual Standard Problems for the actual filling of these arrays.
The most simple option is to call subroutine FILCOF (Section 6.2) in FILNLN.

USER In this array the user must store the real information concerning the coefficients and other relevant parameters.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition.

ISOL In this array information concerning the last computed solution is stored. The essential boundary conditions must be stored in this array, either in the main program, or in the first call of FILNLN.
The user may fill an initial estimate of the solution in the array corresponding to ISOL, for example by subroutine CREATE (5.3.1).

RELAX The user may give RELAX the value of a relaxation parameter ω ($0 \leq \omega \leq 2$) in the iteration process. This parameter is only used when NITER > 0.
When $\omega = 0$ or 1 the solution of the previous iteration $\mathbf{u}_{NITER}$ is taken as new estimate $\mathbf{u}_{old}$ of the solution.
When $\omega \neq 0$ or 1, $\mathbf{u}_{old}$ is taken equal to $\mathbf{u}_{old} = \omega\mathbf{u}_{NITER} + (1-\omega)\mathbf{u}_{NITER-1}$, $\mathbf{u}_{-1} = \mathbf{u}_0$.

DIFFER Subroutine NONLIN stores the difference $\max_{nodes} |\mathbf{u}_{NITER} - \mathbf{u}_{NITER-1}|$ in DIFFER. When NITER=0, DIFFER gets the value infinity.

Input

NITER and DIFFER have been filled by subroutine NONLIN.
The arrays KMESH and KPROB must have been filled before the call of subroutine NONLIN.
The user may have filled the essential boundary conditions and/or an initial estimate in array ISOL.

Output

The user may have filled the boundary conditions and/or an initial estimate in array ISOL by subroutine FILNLN.

The user must fill the arrays IUSER and USER in subroutine FILNLN.
RELAX must have been given a value in FILNLN.

layout

Subroutine FILNLN must be programmed as follows:

```
      subroutine filnln ( niter, iuser, user, kmesh, kprob, isol, relax,
     +                    differ )
      implicit none
      integer niter, iuser(*), kmesh(*), kprob(*), isol(5)
      double precision user(*), relax, differ

      if ( niter .eq. 0 ) then

         statements to fill IUSER and USER
         if necessary statements to fill ISOL

      else

         statements to change IUSER and USER if necessary
         statements to give RELAX a value

      end if

      end
```

10 Solution of eigenvalue problems

10.1 Introduction

In this chapter we describe all SEPRAN subroutines that are available to compute eigenvalues and eigenvectors of matrices. In this way physical eigenvalue problems may be solved.
At this moment only an old fashioned subroutine EIGVAL (10.3).
New subroutines are in preparation.

10.2 General subroutine for the computation of eigenvalues and eigenvectors

This section is not yet available. See Section 10.3 for the only available eigenvalue subroutines. If you need these subroutines please contact SEPRA.

10.3 Old fashioned subroutines to compute eigenvalues and eigenvectors (EIGVAL)

Description

Subroutine EIGVAL computes the smallest n eigenvalues of the generalised eigenvalue problem $\mathbf{Sx} = \lambda\mathbf{Mx}$ by a Lanczos method. Both the stiffness matrix **S** and the mass matrix **M** must be symmetric positive definite.
At this moment only diagonal mass matrices are permitted, hence the subroutine can only be used for eigenvalue problems that are solved by linear, bilinear or trilinear finite elements. For an example of the use of EIGVAL, see Section 25.5.1.

Call

```
CALL EIGVAL( EVAL, NR, IVEC, ISTIFF, INTMT1, IMAS, INTMT2, KPROB, ICHEIG )
```

INTEGER NR, IVEC(5,∗), ISTIFF(5), INTMT1(5), IMAS(5), INTMT2(5), KPROB(∗), ICHEIG(10)

DOUBLE PRECISION EVAL(∗)

EVAL Variable length array of length NR+5. At output EVAL contains the NR smallest eigenvalues of the matrix $\mathbf{M}^{-1}\mathbf{S}$, hence of the generalised eigenvalue problem $\mathbf{Sx} = \lambda\mathbf{Mx}$. The i^{th} eigenvalue is stored in EVAL(5+i), where the eigenvalues are numbered in increasing order. Remember the first position of EVAL must be filled by the user, with the declared length of EVAL.
The subroutine does not recognise multiple eigenvalues. If an eigenvalue has a multiplicity larger than one, then the eigenvalue is stored only once.

NR Number of eigenvalues that must be computed. Only the NR smallest eigenvalues of the problem are stored in EVAL.

IVEC When the eigenvectors must be computed, then information of the eigenvectors corresponding to the NR eigenvalues is stored in IVEC. IVEC is an integer vector of length $5 \times NR$. In contrast with common SEPRAN practice this vector is not treated as a variable array, but must be considered as a concatenation of NR integer SEPRAN arrays of length 5 each. The user must take care of a correct declaration himself.
The storage of the information of the eigenvectors in IVEC is the same as for solution arrays, however, with the special property that the first 5 positions correspond to the first eigenvector, the positions 6 to 10 to the second eigenvector etc. This implies that when the user wants to use IVEC in a plot or print subroutine, he has to use IVEC((i-1) × 5) in the call, if the i^{th} eigenvector is required. See Section 25.5.1 for an example.
Array IVEC is only used when eigenvectors are required. The actual eigenvectors are stored on backing storage (file 3).

ISTIFF Standard SEPRAN array containing information of the stiffness matrix **S**. Output of one of the matrix building subroutines, for example subroutine BUILD (7.2) (Section 7.2).

INTMT1 Standard SEPRAN array, containing information concerning the structure of the matrix **S**. Output of a SEPRAN starting subroutine (Section 4.2) or subroutine COMMAT (4.5.2) (4.5).

At this moment ISTIFF and INTMT1 must correspond to a compact symmetric storage, i.e. METHOD must have the value 5.

IMAS Standard SEPRAN array containing information of the mass matrix **M**. Output of one of the matrix building subroutines, for example subroutine BUILD (Section 7.2).
At this moment IMAS must correspond to a diagonal mass matrix. As a consequence accurate eigenvalues and eigenvectors can only be expected if linear, bilinear or trilinear finite elements

are used.
If subroutine BUILD (7.2) is called for a standard SEPRAN finite element (type numbers $\geq$ 100), then the mass matrix corresponding to that problem is automatically computed.

INTMT2 Array INTMT2 is not used at this moment.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

ICHEIG Input array of length 10 containing some information for the subroutine. All positions that are not used at this moment must be filled with zeros in order to be sure that the subroutine still works with future updates. At this moment only the first 2 positions contain relevant information.
ICHEIG must be filled as follows:

ICHEIG(1): ICHEIGVECTOR

ICHEIG(2): ICHPRINT

ICHEIG(3-10): 0

with

ICHEIGVECTOR Indication whether the eigenvectors must be computed (1) or not (0).

ICHPRINT Indication whether the subroutine must print information concerning the iteration process (1) or not (0)

Input

The arrays ISTIFF, INTMT1, IMAS, ICHEIG and KPROB must have been filled.
NR must have a value.

Output

The computed eigenvalues are stored in array EVAL from position 6.
If the eigenvectors must be computed, then information concerning the eigenvectors is stored in array IVEC.

11 Solution of time-dependent problems

11.1 Introduction

In this chapter we describe all available subroutines for time-dependent problems, except those related to free boundary problems. Those special subroutines are treated in Section 17.5.

At this moment the following subroutines for time-dependent problems are available:

TIMPRB (11.2) General subroutine to solve time-dependent subroutines.

SOLTIM (11.3) Old fashioned subroutine to solve time-dependent subroutines.

FILTIM (11.4) Help subroutine for subroutine SOLTIM. This user written subroutine is used to indicate which differential equations are to be solved, and what processes must take place in each time-step.

11.2 General subroutine for the solution of time-dependent problems

Description

Subroutine TIMPRB is the present general subroutine to solve time-dependent problems, both linear and non-linear. At the users choice the time-step is computed automatically, or is given by the user.
The method of lines is used for the solution of the time-dependent partial differential equations, that is first the differential equation is discretized in space, and then the resulting system of ordinary differential equations is solved by some classical integration method.

Heading

```
subroutine timprb ( kmesh, kprob, intmat, isol, iintim )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), INTMAT(5), ISOL(5,∗), IINTIM(∗)

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

INTMAT Integer array containing information about the matrix structure. Output of a SEPRAN starting subroutine for example SEPSTN (4.2.3) or subroutine COMMAT (4.5).
In general the matrix to be decomposed is non-symmetrical, and METHOD in the input of SEPSTN (4.2.3) or JMETOD in COMMAT (4.5.2) should be equal to 2 OR 6, in some special cases JMETOD = 1 OR 5, may be used.
For a proper choice in the case of standard problems, consult the manual "Standard Problems".
If the user solves more than one problem in one run, he indicates the type of problem to be solved by SOLTIM, by using the proper choice of array INTMAT. Hence, array INTMAT indicates which problem must be solved.

ISOL Array containing information of the solution vector and other related vectors. At input the initial condition must be filled in the solution vector corresponding to ISOL. At output this vector contains the solution at the end time.

IINTIM Input array defined by the user.
With this array the user may indicate which actions he wants to be performed by this subroutine.
IINTIM must be filled as follows:

Pos. 1: Last entry number that has been filled by the user. If 0 or 1 only defaults are used

Pos. 2: IREAD Defines how the input must be read.
Possible values:

0: All SEPRAN input has been read by subroutine SEPSTN (4.2.3) until `end_of_sepran_input` or end of file has been found. It concerns the input described in the input block `time_integration`, see the User Manual Section 3.2.15.
This input is used.
The user may put this input into IBUFFR/BUFFER himself using subroutine FILLINP (4.7). For a description of the input, see the part: Putting the input into IBUFFR/BUFFER, without actual reading from the input file at the end of this section.

1: The input is read from the standard input file.
2: The last input read as described for TIMPRB is used.
Default value: 0.

3: ISEQNR, This parameter is only used if IREAD=0. It indicates the sequence number of the input for the time integration to be used.
Default value: 1.

4: IVECTR, sequence number of solution vector with respect to array ISOL. Hence the initial condition is supposed to be stored in the array corresponding to ISOL(.,IVECTR). The computed solution is also stored in this array.
Default value: 1.

5: MAXVEC, Maximum number of vectors stored in isol.
Default value: 1.

Input

The arrays KMESH, KPROB, INTMAT, IINTIM and ISOL must have been filled.

Output

The solution vector corresponding to ISOL(.,IVECTR) has been filled with the solution at the end time TEND.

Warning

The integration in time of a system of ordinary differential equations can cause many problems. If the user wants to be sure that the solution obtained is reliable, he can make the following tests:

- compare results of different methods
- compare results obtained using different step-sizes TSTEP for the same method
- compare results obtained using different accuracies EPSUSR for the same method.

If the user is not sure which method should be used in his case, he is advised to use ICHTIM = 1, and to experiment with several values for EPSUSR.

Putting the input into IBUFFR/BUFFER, without actual reading from the input file

The input for the subroutine is usually read from the input file as described in Section 3.2.15 of the Users Manual. However, sometimes the user may want to fill this input himself into IBUFFR and BUFFER and address it by the sequence number IREAD (IINTIM(2)).
In that case subroutine FILLINP (4.7 may be used with `isubr` = 15 and `iseqnr` equal to IREAD (>0). The user must create to input arrays: IINTIM_READ (IINPUT) and RINTIM_READ (RINPUT).
These arrays must be filled as follows:

IINTIM_READ Integer array where the length is implicitly defined by the input.
IINTIM_READ must be filled as follows:

Pos. 1: Last entry number that has been filled by the user. If 0 or 1 only defaults are used.

Pos. 2: ichtim: Type of integration method
Possible values:
1: Euler/Trapezoid rule with self-selecting step-size
2: Rozenbrock/Wanner rule with self-selecting step-size

3: Runge-Kutta-Fehlberg (1-2 order) with self-selecting step-size
4: Runge-Kutta-Fehlberg (3-4 order) with self-selecting step-size
5: Euler implicit, time-step given by the user
6: Implicit trapezoid rule, time-step given by the user
7: Euler explicit, time-step given by the user
8: Runge-Kutta 4 explicit, time-step given by the user
9: Implicit (modified) theta method ($0 < \theta <= 1$), θ and time-step given by the user (includes 5,6)
10: Fractional step method
11: Central differences for second order time-derivatives, time-step given by the user.
12: Generalized theta method

Pos. 3: linear: Indicates if the process must be considered as linear, i.e. with linearized coefficients (0) or as strictly non-linear (1), which implies that in each time step iterations are carried out.

Pos. 4: ifiltm: Indicates if the user builds the matrix and right-hand side by a call to filtim (1) or not (0)

Pos. 5: ncopeq: Number of coupled problems to be solved The Silechi approach is used, i.e. the already computed unknowns are used for the computation of the coefficients of the following equations in the same time-step

Pos. 6: itimbc: Indicates how often boundary conditions must be computed. Possible values:
0: Boundary conditions are time-dependent and must be filled each step
1: Boundary conditions are time-independent and must be filled only once
2: Boundary conditions are not filled by this subroutine,
but by subroutine filtim
Only in combination with ifiltm=1

Pos. 7: ireadbc: Indicates how boundary conditions must be computed. This parameter is only used if itimbc<2.
Possible values:
0: Read input from the standard input file during the first time step
-1: Copy boundary conditions from array islold.
Only if itimbc=1
>0: The input has been read by SEPSTN.
ireadbc indicates the corresponding sequence number.
The corresponding input may be stored in IBUFFR/BUFFER by subroutine FILLINP (4.7) using `isubr` = 5.

Pos. 8: itimcf: Indicates how coefficients are treated.
Possible values:
0: Coefficients are filled by the user in the arrays iuser and user. This may be done in subroutine filtim. The parameter ireadcf is not used.
1 Coefficients are not filled in iuser and user but in the buffer array instead. The parameter ireadcf is used.
This means that they can only be used inside this subroutine.
The corresponding input may be stored in IBUFFR/BUFFER by subroutine FILLINP (4.7) using `isubr` = 6.

Pos. 9: iseqfr: Sequence number of first vector with respect to array islold. It is supposed that the unknown vectors to be solved are iseqfr, iseqfr+1, ... , iseqfr+ncopeq-1 in that sequence

Pos. 10: iconsstiff: If 1 the stiffness matrix is constant, otherwise 0

Pos. 11: iconsmass: If 1 the mass matrix is constant, otherwise 0

Pos. 12: iconsrhs: Information concerning the right-hand side i.e. the function f (no effect of boundary conditions).
- 0: The right-hand side is time-dependent
- 1: The right-hand side is time-independent
- 2: The right-hand side is zero

Pos. 13: ireadsl: Parameter with respect to the reading of information for subroutine solvel.

The corresponding input may be stored in IBUFFR/BUFFER by subroutine FILLINP (4.7) using `isubr` = 8.
If ireadsl = -1, only defaults are used.

Pos. 14: imas: Parameter imas for building of mass matrix

Pos. 15: numold: Number of vectors stored in array islold

Pos. 16: isubdivide: indicates if quadratic elements must be treated as a conglomerate of linear elements (1) or as quadratic elements itself (0)

Pos. 17: nfrac: Number of fraction steps

Pos. 18: ndt: Number of different time steps

Pos. 19: icall_outtim: defines whether OUTTIM must be called (1) or not (0)

Pos. 20: seq_output, defines the sequence number for the input definition of OUTPUT. The corresponding input may be stored in IBUFFR/BUFFER by subroutine FILLINP (4.7) using `isubr` = 9.

Pos. 21: iretur: Indicates if the subroutine must stop if no convergence has been found (0) or that control must be given back to the main program (1) Only if stationary solution is required

Pos. 22: iprint Defines the amount of output produced in case of a stationary solution. Possible values:
- -1: All output information is suppressed, even the message of no-convergence
- 0: No information about the iteration process is printed except in the case of an error
- 1: The number of iterations performed is printed
- 2: Some extra information about the iteration process is printed for each iteration
- 3: See 2, in this case also the ratio of two succeeding iterations is printed
- 100: Contains the maximum of preceding values and also prints the solution in each iteration

Pos. 23: IUSE_CTIMEN Defines if the input with respect to the time step, initial time and end time is given in common CTIMEN (1) or in RINTIM (0).

Pos. 24: KEEP_t Defines what happens with the time t during a time step.
Possible values:
- 0 Standard situation, the time is raised in each time step.
- 1 The time is not changed during the time step.
  Corresponds to the option `keep_t` in the input file.
- 2 First the time is decreased by one time step, then the computation is performed and after that the time is increased with a time step.
  Corresponds to the option `reuse_time_parameters` in the input file.

Pos. 25: NHIST Number of points in which the time history must be printed
The co-ordinates of the points must be stored in rintim(41)-rintim(40+NHIST×ndim).

Pos. 26: If > 0, it contains the starting address of extra information concerning the individual equations Denote this starting address with istart_info_eq

Pos. 27: maxvec, maximum number of vectors that may be stored in islold

Pos. 28: maxiter: maximum number of iterations in case of a non- linear time-dependent method

Pos. 29: If 1 the solution is supposed to be stationary and it is checked if a stationary solution has been found using rintim(8)/(9).
If 0, then no stationary solution is required

Pos. 30: Indicates if a standard computation is carried out (0) or that only the time is raised and no computation is carried out (1).

Pos. 31-30+ncopeq: May be considered as an array ireadcf with length ncopeq.
This array contains the parameters iread for the coefficients.
ireadcf(j): Indicates how coefficients must be read
Possible values:

0: Read input from the standard input file during the first time step
Each problem requires his own input block.

>0: The input has been read by SEPSTN

If iintim(26)>0, this is followed by information per equation For each equation at least 25 positions are used:
The contents of these 25 positions are as follows:

1: iconsstiff for equation
2: iconsmass for equation
3: iconsrhs for equation
4: ireadsl for equation
5: ireadbc for equation
6: isubdivide for equation
7: imas for equation
8: vector sequence number for equation (Relative to iseqfr)
9: itimbc for equation
10: Starting address of extra computations that must be performed for this equation
If 0, no extra computations must be performed
If > 0 the storage of the extra part is like in to0094: (recursively)

Pos. 1 Contains the type of action Possible values:

-1: last action
8: call to deriv

Depending on the type of action extra positions are needed.
In case of DERIV the positions 2 - 6 are used:

2: Sequence number coefficients
3: IPROB
4: IVECTOR Sequence number output vector
5: ISEQNR Sequence number derivatives
6: ISCALAR Sequence number scalar (makes no sense)

11: limit for equation.
Possible values

0 none
1 minmax

12-25: not yet used

RINTIM_READ Double precision array where the length is implicitly defined by the input.
RINTIM_READ must be filled as follows:

Pos. 1: eps: accuracy for time integration in case of a self-selecting step size

Pos. 2: eps_solv: accuracy for linear solver if an iterative method is used and inpsol must be filled

Pos. 3: t: actual time

Pos. 4: $t0$: initial time

Pos. 5: toutinit: initial time for output with outtim

Pos. 6: toutend: end time for output with outtim

Pos. 7: toutstep: time step for output with outtim

Pos. 8: absaccuracy: Absolute criterion for stationary solution or non-linear iteration

Pos. 9: relaccuracy: Absolute criterion for stationary solution or non-linear iteration

Pos. 10: tthreshold: Threshold time. Divergence of the stationary solution is only checked when $t >$ tthreshold

Pos. 11-20: nfrac thetas

Pos. 21-30: ndt end times

Pos. 31-40: ndt time steps

Pos. 41-40+NHIST*ndim: co-ordinates of points for which the time history must be printed.
Actually the result in the nodes closest to these co-ordinates are printed

11.3 Old fashioned subroutines to solve time-dependent problems (Subroutine SOLTIM)

Description

Subroutine SOLTIM may be used to solve time-dependent problems, both linear and non-linear. At the users choice the time-step is computed automatically, or is given by the user.
The method of lines is used for the solution of the time-dependent partial differential equations, that is first the differential equation is discretized in space, and then the resulting system of ordinary differential equations is solved by some classical integration method.
The task of subroutine SOLTIM is to perform the time-discretization, the space discretization must be performed by the user with help of subroutine FILTIM (Section 11.4)

Call

```
CALL SOLTIM ( ITYTIM, ICHTIM, KMESH, KPROB, INTMAT, ISOL, EPSUSR, USER, IUSER )
```

INTEGER ITYTIM, ICHTIM, KMESH(∗), KPROB(∗), INTMAT(5), ISOL(5), IUSER(∗)

DOUBLE PRECISION EPSUSR, USER(∗)

ITYTIM Choice parameter indicating what type of problem is considered. Possibilities:

1 the problem is linear.

2 the problem is non-linear.
This possibility is not yet available.

ICHTIM Choice parameter for the type of numerical integration to be used in time. See standard text books for the nomenclature. Possibilities:

1 Crank-Nicolson type integration with self-selecting step-size. (implicit method, second order accurate). First order time-derivatives only.

2 Not yet available.

3 Runge-Kutta-Fehlberg type integration with self-selecting step-size. (explicit method, second order accurate). First order time-derivatives only.

4 Not yet available.

5 Implicit first order Euler integration method. The step-size is selected by the user. First order time-derivatives only.

6 Implicit second order trapezoid rule (Crank-Nicolson). The step-size is selected by the user. First order time-derivatives only.

7 Explicit first order Euler method. The step-size is selected by the user. First order time-derivatives only.

8 Not yet available.

11 The system of time-dependent equations to be solved is supposed to contain only second derivatives of time and no first order time-derivatives. Hence:

$$\mathbf{M}\frac{\partial^2 \mathbf{u}}{\partial t^2} + \mathbf{S}\mathbf{u} = \mathbf{F} \tag{11.3.1}$$

The time-derivatives are discretized by a central difference scheme. The other terms are treated explicitly hence:

$$\mathbf{M}\frac{\mathbf{u}^{n+1} - 2\mathbf{u}^n + \mathbf{u}^{n-1}}{\Delta t^2} + \mathbf{S}\mathbf{u}^n = \mathbf{F}^n \tag{11.3.2}$$

The step-size must be given by the user. Also the user must give T0 a value. In ISOL not only information of the initial condition but also of the time-derivative at $t = t^0$

must be given. See ISOL.
In this special case the user must store the boundary conditions at time-level T+TSTEP in array ISOL in subroutine FILTIM.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

INTMAT Integer array containing information about the matrix structure. Output of a SEPRAN starting subroutine for example SEPSTN (Section 4.2) or subroutine COMMAT (4.5).
In general the matrix to be decomposed is non-symmetrical, and METHOD in the input of SEPSTN (4.2.3) or JMETOD in COMMAT (4.5.2) should be equal to 2, in some special cases JMETOD = 1, may be used.
For a proper choice in the case of standard problems, consult the manual "Standard Problems".
If the user solves more than one problem in one run, he indicates the type of problem to be solved by SOLTIM, by using the proper choice of array INTMAT. Hence, array INTMAT indicates which problem must be solved.

ISOL Array containing information of the solution vector. At input the initial condition must be filled in the solution vector corresponding to ISOL. At output this vector contains the solution at the end time.
When the user wants to transport old solution arrays (or arrays corresponding to derived quantities) to the element subroutines, he has to use subroutine BUILD (7.2) in his own subroutine FILTIM.
These old solution arrays may be stored in array ISOL in the same way as array ISLOLD in BUILD (7.2) is used. However, it is necessary that the first five positions of ISOL contain the reference to the initial condition. So, if the user wants to store two arrays ISOL1 and ISOL2 in ISOL, he has to define ISOL as a two-dimensional array of size 5×3 (ISOL(1:5,1:3)), and the contents of ISOL1 and ISOL2 must be copied in ISOL in the following way:

```
                DO ...  k = 1,5
                   ISOL(k,2) = ISOL1(k)
                   ISOL(k,3) = ISOL2(k)
...             CONTINUE
```

Array ISOL is transported to subroutine FILTIM, where it gets the name ISLOLD.
If an equation with second order time-derivatives is solved, the user must give both the initial field ($\mathbf{u}(t = t^0)$) and the initial time-derivative field ($\frac{\partial \mathbf{u}}{\partial t}(t = t^0)$). At $t = t^0$ information of the initial condition must be stored into ISOL(k,1) (k=1,...,5) and information of the time-derivative in ISOL(k,2) (k=1,...,5). At output ISOL(k,1) refers to the solution at T = TOUT ($t = t^n$) and ISOL(k,2) to the solution at $t = t^{n-1}$. At a restart ($t > t^0$) it is necessary to give the solution at $t = t^n$ and at $t = t^{n-1}$. So the output at T = TOUT must be used as input for the restart.
In subroutine FILTIM the user must fill the actual boundary conditions in array ISOL, unless these boundary conditions are time-independent. For the cases ICHTIM = 1, . . . , 10, these boundary conditions must be filled at time-level T, whereas ICHTIM = 11 demands boundary conditions at the new time-level T+TSTEP.

EPSUSR When the user chooses one of the methods with self-selecting step-size
(ICHTIM = 1 to 4) he must give EPSUSR the accuracy required for the solution, otherwise EPSUSR may have any value.
It is in any case necessary that EPSUSR has been initialized before the call of SOLTIM (for example by 0).

USER,IUSER In these arrays information concerning the coefficients etc. must be stored. Consult the manual "Standard Problems" for the actual filling of these arrays.
When the user uses elements with type numbers smaller than 99, he may fill IUSER and USER in his own way.
IUSER and USER are variable length arrays and should be dimensioned properly. Their first positions must be filled by the user.

Common Blocks

Besides the parameters in the parameter list, SOLTIM uses also the common block CTIMEN to control the time integration:

```
COMMON /CTIMEN/ T, TOUT, TSTEP, TEND, T0, RTIME(5), IFLAG, ICONS, ITIMDEP, ITIME(7)
```

Common block CTIMEN must be submitted by the user to the main program. Meaning of the parameters in CTIMEN:

T Actual time. At input (in the main program) the user should set T equal to the initial time.

TOUT The integration is performed from initial time to TOUT. When TOUT is reached control is returned to the main program. The user must give TOUT a value in the main program.

TSTEP Time step for the numerical integration. The time-step may be chosen by the user (ICHTIM = 5 to 8) or by the subroutine. The user may change his own chosen time-step in subroutine FILTIM.

TEND End time of integration. When TEND is reached control is returned to the main program, but also all memory used by SOLTIM is released. Restart is not possible for t > TEND. The user must give TEND a value in the main program.

T0 Initial time for the first integration step. T0 = t^0. The user must give T0 a value before the first call of SOLTIM in a series of calls of SOLTIM.

RTIME Dummy real array of length 5. This array is not yet used, but must be positioned in the common block

IFLAG Error indicator. When IFLAG = 3 at the output, the process has been terminated succesfully; if IFLAG = 4 the integration process has been stopped due to errors or inaccuracy. This may for example be the case if a discontinuity is present in the solution.

ICONS Indication if the matrices **M** and **S** as well as the time-step TSTEP are constant (ICONS=1) or not (ICONS=0). ICONS=1 might result in a faster program in the case of a constant step-size algorithm.

ITIMDEP This parameter is used to indicate if a the solution is time-dependent (1) or not (0). This is the only parameter that gets a value by the SEPRAN starting subroutines. Subroutine SOLTIM changes this value to 1.

ITIME Dummy integer array of length 7. This array is not yet used, but must be positioned in the common block

Mark that the parameters in CTIMEN are not initialized by SEPRAN subroutines, except ITIMDEP. However, some of the parameters may get a value by subroutine SOLTIM. All other ones must be initialized by the user himself! *Input*

The parameters ITYTIM, ICHTIM and EPSUSR must have a value.
The parameters T, TOUT, TEND, T0 and if necessary TSTEP must have a value.
The arrays KMESH, KPROB, INTMAT and ISOL must have been filled.
If necessary the arrays IUSER and USER must have been filled.

Output

The solution vector corresponding to ISOL has been filled with the solution at time TEND. For second order time-derivative problems also the solution at $t = t^{n-1}$ has been stored in ISOL. Depending of the type of method (self-selecting step-size), IFLAG has got a value.

Subroutines called

During the integration process, the user-written subroutine FILTIM is called frequently. In this subroutine the user takes care of the filling of:

- the mass-matrix **M**
- the stiffness-matrix **S**
- the right-hand-side vector **F**

all in accordance with the prescribed boundary conditions. For a description of how subroutine FILTIM must be programmed, see Section 11.4.

Warning

The integration in time of a system of ordinary differential equations can cause many problems. If the user wants to be sure that the solution obtained is reliable, he can make the following tests:

- compare results of different methods
- compare results obtained using different step-sizes TSTEP for the same method
- compare results obtained using different accuracies EPSUSR for the same method.

If the user is not sure which method should be used in his case, he is advised to use ICHTIM = 1, and to experiment with several values for EPSUSR.

11.4 Subroutine FILTIM

Description

This user-written subroutine is called by subroutine SOLTIM during the execution of the time-integration. In this subroutine the user has to fill the following quantities (at time-level t):

- the mass matrix **M**
- the stiffness matrix **S**
- the right-hand-side vector **F**

The effect of the essential boundary conditions must not be stored in the right-hand-side vector. Furthermore in the case of time-dependent essential boundary conditions, the user must fill the essential boundary conditions in array ISOL at time-level T (i.e. actual time-level) in the case ICHTIM = 1 to 10 and at time-level T+TSTEP, i.e. the new time-level $t^{n+1} = t + \Delta t$ in the case ICHTIM = 11.

Call

```
CALL FILTIM ( KMESH, KPROB, INTMAT, ISOL, MATRST, MATRMS, IRHSD,
              ISLOLD, USER, IUSER )
```

INTEGER KMESH(∗), KPROB(∗), INTMAT(5), ISOL(5), MATRST(5), MATRMS(5), IRHSD, ISLOLD(5), IUSER(∗)

DOUBLE PRECISION USER(∗)

KMESH, KPROB, INTMAT See subroutine SOLTIM.

ISOL In this array the solution will be stored. The user must fill the essential boundary conditions at time-level T or T+TSTEP in ISOL.

MATRST In this array, the user must store the stiffness matrix.

MATRMS In this array the user must store the mass matrix.

IRHSD The right-hand-side vector must be stored by the user in IRHSD. No effect of boundary conditions must be taken into account.

ISLOLD Solution at preceding time-level. If the user did store more than one array in ISOL in the call of SOLTIM, then these arrays are stored in array ISLOLD. So in this way ISLOLD may be used in the call of subroutine BUILD (7.2) to transport old solutions to the element subroutines. This array may be used in non-linear cases.

IUSER, USER Communication arrays. These arrays are given in the heading of subroutine SOLTIM. Subroutine SOLTIM does not use these arrays, hence the user may fill them for his own use for example to store coefficients.

Common Blocks

See subroutine SOLTIM

Input

Subroutine SOLTIM fills array ISLOLD as well as the parameters T, TSTEP and IFLAG in common block CTIMEN. TSTEP is only filled when a self selecting step-size subroutine is used.

Output

The arrays MATRST, MATRMS and IRHSD must have been filled by the user. The essential boundary conditions at time-level T must have been stored in array ISOL.

Layout

Subroutine FILTIM must be programmed as follows:

```
      subroutine filtim ( kmesh, kprob, intmat, isol, matrst, matrms,
     +                    irhsd, islold, user, iuser )
      implicit none
      integer   kmesh(*), kprob(*), intmat(*), iuser(*), user(*),
     +          isol(*), irhsd(*), matrst(*), matrms(*), islold(5,*)
      double precision t, tout, tstep, tend, t0, rtime
      integer iflag, itime, icons
      common /ctimen/ t, tout, tstep, tend, t0, rtime(5), iflag, icons,
     +                itime(8)


            statements to put the essential boundary conditions
            in array isol at time-level t + tstep
            statements to fill matrst, matrms, and irhsd

      end
```

Remark

The mass-matrix **M** may be either a diagonal matrix or a band matrix. If the matrix is diagonal, it can be stored as a right-hand-side vector. In that case subroutine BUILD (7.2) may be used to compute both matrices and the right-hand side in one call. When the mass matrix is not a diagonal matrix, subroutine BUILD (7.2) has to be used. This subroutine must be called twice, once for the construction of stiffness matrix and right-hand side, and once for the construction of the mass-matrix. In the first call of BUILD (7.2), JCHOIS should have the value 11, in the second call the value 14. See subroutine BUILD (7.2).
For an example of a time-integration program using SOLTIM and FILTIM, see 25.4.2 (diagonal matrix **M**) and 25.4.3 (non-diagonal mass matrix). 25.4.4 treats the solution of a second order time-derivative equation using SOLTIM and FILTIM.

12 Manipulation of SEPRAN vectors and matrices

12.1 Introduction

In this section we shall describe some subroutines that may be used to manipulate SEPRAN vectors or matrices. It concerns specifically arrays that have the structure of a SEPRAN solution vector or of a vector of special structure. Since all SEPRAN vectors are hidden in the large buffer array, direct manipulation is not possible. It is necessary to use the memory management subroutines described in Chapter 21 or to use the subroutines described in this chapter. The result of these manipulations is again a SEPRAN vector with one of the structures SEPRAN recognizes.

In this chapter we deal with the following paragraphs:

12.2 treats subroutine MANVEC, a general vector manipulation subroutine that should be sufficient for most problems.

12.3 deals with a number of user function subroutines that may be called by subroutine MANVEC depending on the input parameters.

12.4 treats some subroutines that are used in previous versions of SEPRAN and whose task is now carried out by MANVEC.

12.5 is devoted to copying of SEPRAN solution vectors and vectors of special structure (subroutines COPYVEC and COPYVC)

12.6 treats the addition of Sepran matrices (subroutines ADDMAT and ADDMT1)

12.7 deals with copying of SEPRAN matrices.

12.8 is devoted to matrix vector multiplication.
This is of special interest for the solution of time-dependent equations.

12.9 describes subroutine LOCTRN, which enables the user to perform the local transformations on a vector and also the back-transformations.

12.10 describes subroutine PRINCIPSTR, which is used to compute the principal stresses once the stress tensor has been computed.

12.11 describes subroutine DELSEPAR, which is used to delete SEPRAN arrays that have been created before.

12.2 Manipulation of vectors (subroutine MANVEC)

At this moment there are two SEPRAN subroutines that may be used to manipulate SEPRAN vectors.
These subroutines are:

MANVEC (12.2.1) Copy a complete SEPRAN vector into another one.

MANVECBF (12.2.2) Same as MANVEC, however, with extra parameters IBUFFR and BUFFER.

12.2.1 subroutine MANVEC

Description

In order to perform simple algebraic manipulations with SEPRAN vectors like taking the inner product, component-wise multiplication of two vectors and so on, it is necessary to have these arrays immediately available. However, since all SEPRAN arrays are stored in the large buffer array direct access of these arrays is not possible. An extra complication is that for example the solution arrays may have been renumbered internally. So manipulation is only possible if the structure of the arrays is known as well as the positions in the buffer array.
The memory management subroutines described in Chapter 22 make it possible to enter specific arrays. The structure of the arrays is described in Chapter 23.
However, for simple manipulations subroutine MANVEC has been developed, which automatically performs a number of operations on SEPRAN vectors, returning either a constant or another SEPRAN array. Due to this subroutine the user does not have to know anything about the internal structure of the SEPRAN arrays.

Heading

```
subroutine manvec ( iinvec, rinvec, invec1, invec2, iresvc, kmesh, kprob )
```

Parameters

INTEGER IINVEC(∗), INVEC1(5,∗), INVEC2(5), IRESVC(5), KMESH(∗), KPROB(∗)

DOUBLE PRECISION RINVEC(∗)

IINVEC In this integer input array it is indicated what type of manipulations must be performed by subroutine MANVEC. For some cases IINVEC acts both as input and as output array. The user himself is responsible for the declaration of IINVEC. SEPRAN has no possibility of checking the declared size of this array, so the user must be very careful in order not to overwrite parts of the memory and in that way produce unexpected results. In almost any problem a length of 10 for IINVEC suffices.
The entries of IINVEC must be filled by the user as follows:

1: LAST Number of entries of IINVEC that are filled by the user. IINVEC(1) must be at least equal to 2. For all positions beyond position IINVEC(1), defaults are used. If a user fills an entry of IINVEC with a zero, also the default value is used.

2: ICHOICE General choice parameter indicating what type of algebraic manipulations have to be carried out. ICHOICE must always have been given a value by the user.
First we give a short description of the possible values of ICHOICE and their meaning. The extended description is given under the heading *extended description of the various possibilities*.

1-25: Compute constant quantities from vectors.

26-50: Compute new vectors from old vectors.

> 50: Map a vector onto another vector.

1 Average: $c = average\ (\mathbf{u}_1)$.

2 Extract value of one degree of freedom in one point from $\mathbf{u}_1$.

3 Minimum and maximum value of $\mathbf{u}_1$ or abs($\mathbf{u}_1$).

4 Inner product: $c = (\mathbf{u}_1, \mathbf{u}_2)$.

5 Norm of $\mathbf{u}_1$ or $\mathbf{u}_1 - \mathbf{u}_2$: $\| \mathbf{u}_1 \|$ or $\| \mathbf{u}_1 - \mathbf{u}_2 \|$:

6-25 Not yet available.

26 Subtraction of constant: $\mathbf{u}_{out} = \mathbf{u}_1 - c$.

27 Linear combination: $\mathbf{u}_{out} = c_1\ \mathbf{u}_1 + c_2\ \mathbf{u}_2$.

28 Put value of one degree of freedom in one point into $\mathbf{u}_{out}$.

29 Modulus of complex vector.

30 Phase of complex vector.

31 User defined function of two SEPRAN vectors: $\mathbf{u}_{out} = \mathbf{f}(\mathbf{u}_1, \mathbf{u}_2)$.

32 User defined function of n SEPRAN vectors and the co-ordinates:
$\mathbf{u}_{out} = \mathbf{f}(\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n, \mathbf{x})$.

33 Length of vector composed by components:
$u_{out}(i) = (|u_1(i)|^2 + |u_2(i)|^2)^{\frac{1}{2}}$.
or
$u_{out}(i) = (|u_1(i)|^2 + |u_2(i)|^2 + |u_3(i)|^2)^{\frac{1}{2}}$.

34 Complex conjugate of complex vector.

35 SEPRAN function FUNADA of two SEPRAN vectors: $\mathbf{u}_{out} = \mathbf{FUNADA}(\mathbf{u}_1, \mathbf{u}_2)$.

36 Vector consisting of exactly one of the degrees of freedom from the input vector. Hence one of the components of a vector quantity is extracted.

37 Real part of complex vector.

38 Imaginary part of complex vector.

39 Linear combination: $\mathbf{u}_{out} = c_1\ \mathbf{u}_1 + c_2\ \mathbf{u}_2$, with the restriction that $\mathbf{u}_{out} \geq c_3$.

40 Component-wise product of two vectors: $\mathbf{u}_{out}(i) = c_1\ \mathbf{u}_1(i) \times \mathbf{u}_2(i)$.

41 Point-wise inner-product of two vectors: $\mathbf{u}_{out}(i) = \sum_j \mathbf{u}_1(i)_j + \mathbf{u}_2(i)_j$.

42 Limit u to values between rinvec(1) and rinvec(2)

43 uout is redefined function of vec1.
$\mathbf{u}_{out} = \mathbf{f}(\mathbf{u}_1)$.

44 uout(i) is a vector with 1 component in the points where vec1(i,.) (corresponding to INVEC1) is defined. vec1 must be a vector with ndim components per point. uout(i) = $\mathbf{a} \cdot$ vec1(i).
The vector **a** must be stored in rinvec(1..ndim).
The type of output vector must be defined in ISPEC (0=solution vector, >0 vector of special structure).
The vector **a** must be stored in RINVEC(1-3).

45 uout(i) is vector with ndim components in the points where vec1(i) (corresponding to INVEC1) is defined.
vec1 must be a vector with 1 component.
uout(i) = vec1(i) **a**.
The vector **a** must be stored in rinvec(1..ndim).
The type of output vector must be defined in ISPEC (0=solution vector, >0 vector of special structure).
The vector **a** must be stored in RINVEC(1-3).

46 **uout = u - u · n n**.
The input vector must be a vector with ndim components defined on a curve (2d) or a surface (3d) only.
The normal on this curve or surface is computed and the normal component of **u** in the normal direction **n** is subtracted from **u**, thus making **u** a vector in the curve or surface only.
ISPEC must contain the curve or surface number.

51 Map a SEPRAN array of special structure or of type solution array onto another SEPRAN array of special structure or of type solution array. The arrays may belong to different problems.

52 Map a SEPRAN array vector of type 110 or 115 onto vector of type 116, i.e. map a vector defined per node onto a vector defined per element. Line elements and connection elements are skipped. At this moment the mapping is restricted to meshes consisting of linear triangles only. The number of degrees of freedom per point in the array to be mapped must be constant.

3: IDEGFD Depending on the value of IDEGFD, the algebraic manipulations will be performed with respect to all degrees of freedom of the vectors (IDEGFD = 0), with all internal degrees of freedom i.e. not with the essential boundary conditions
(IDEGFD = -1), or with the $IDEGFD^{th}$ degree of freedom only (IDEGFD > 0).
Default value: IDEGFD = 0.

4: IPOINT Nodal point number. See extended description for its use.
Default value: IPOINT = 1.

5: ISPEC The use of ISPEC depends on the VALUE of ICHOICE.
See *extended description of the various possibilities*.
Default value: Depending on ICHOICE.

6: ICHNRM The use of ICHNRM depends on the VALUE of ICHOICE. See *extended description of the various possibilities*.
Default value: ICHNRM = 3.

7: ISKIP Depending on the value of ISKIP element groups must be skipped in the computation (ISKIP > 9) or not (ISKIP=0).
If ISKIP > 9, the value of ISKIP denotes the starting position in array IINVEC where information concerning the elements to be skipped is stored. In that case it is necessary that the user has filled the positions IINVEC(ISKIP) , . . . , ISKIP(ISKIP+*nelgrp*-1), where *nelgrp* denotes the number of element groups. Only element groups defined in the mesh are taken into account, so boundary element groups are not considered as special class. ISKIP in the range 1 < ISKIP < 10 is not allowed. If IINVEC(ISKIP+*ielgrp*-1) = 0, element group *ielgrp* must not be skipped, if IINVEC(ISKIP+*ielgrp*-1) = 1, the element group must be skipped.
Default value: ISKIP = 0.

8: IDEGFD1 First degree of freedom in a point..
Default value: 1.

9: IDEGFD2 Second degree of freedom in a point.
Default value: 2.

10: IDEGFD3 Third degree of freedom in a point.
Default value: 3.

11: IPROB Problem number. This number is only used for the output vector.
Default value: 1.

The other positions of IINVEC depend on ICHOICE and are described in the part extended description.

RINVEC Double precision input and output array. The contents of RINVEC depend on the contents of IINVEC. See the *extended description of the various possibilities*.
If RINVEC contains only complex quantities then either RINVEC must be defined as a (double precision) complex array or it must be used as a real array, where each complex quantity is stored in two subsequent positions. So in that case the i^{th} parameter is stored in RINVEC(2×i-1) and RINVEC(2×i).

INVEC1 Integer SEPRAN array of length 5 corresponding to a solution vector or a vector of special structure. In the *extended description of the various possibilities* the real (or complex) vector corresponding to INVEC1 is denoted by $\mathbf{u}_1$.
For some possibilities of ICHOICE, $\mathbf{u}_1$ is not only an input vector but may be also output vector.
In some cases INVEC1 corresponds to more than one vector ($\mathbf{u}_1$, $\mathbf{u}_2$, ..., $\mathbf{u}_n$). In that case INVEC1 is supposed to be a two-dimensional array of size $5 \times \alpha$ with $\alpha \geq n$: INVEC1(1:5,1:α).
In INVEC1(i,j), (i=1, 2, ..., 5) information about the j^{th} input vector must be stored
(1$\leq$ j $\leq$ n). This may be done for example by copying the contents of other SEPRAN arrays into INVEC1. Suppose for example that 4 SEPRAN vectors ISOL1, ISOL2, ISOL3 and ISOL4 must be stored into INVEC1 in that sequence. Then the following statements may be applied:

```
      DO 100  i = 1, 5
          INVEC1(i,1) = ISOL1(i)
          INVEC1(i,2) = ISOL2(i)
          INVEC1(i,3) = ISOL3(i)
          INVEC1(i,4) = ISOL4(i)
100   CONTINUE
```

See the *extended description of the various possibilities* for the description of INVEC1.

INVEC2 Integer SEPRAN array of length 5 corresponding to a solution vector or a vector of special structure. In the *extended description of the various possibilities* the real (or complex) vector corresponding to INVEC2 is denoted by $\mathbf{u}_2$.
See the *extended description of the various possibilities* for the description of INVEC2.
Restriction: At this moment INVEC2 must have exactly the same structure as INVEC1 if it is used. The corresponding arrays must be both real or both complex.

IRESVC Integer SEPRAN array of length 5 in which the output of the algebraic manipulation will be stored. In the *extended description of the various possibilities* the real (or complex) vector corresponding to IRESVC is denoted by $\mathbf{u}_{out}$.
See the *extended description of the various possibilities* for the description of IRESVC.
In $\mathbf{u}_{out}$ only the $IDEGFD^{th}$ degree of freedom in the nodal points gets a value when IDEGFD > 0. The other degrees of freedom are copied from $\mathbf{u}_1$.

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

KPROB Standard SEPRAN array containing information about the problem description. KPROB must have been filled before for example by a SEPRAN start routine that reads the mesh.

Input

Array IINVEC must have been filled, at least the positions 1 and 2.
The arrays KMESH, KPROB, INVEC1, and if necessary INVEC2 must have been filled.

For some problems array RINVEC must also have been filled.

Output

Depending on the value of ICHOICE the arrays INVEC3 and/or RINVEC have been filled or changed.

Extended description of the various possibilities

In this section the input and output of MANVEC is described as function of the parameter ICHOICE (IINVEC(2)).

1 The average value of the vector $\mathbf{u}_1$ is computed and stored in RINVEC(1). If $\mathbf{u}_1$ is complex also the average value is complex.
At output IINVEC(5) is equal to the total number of degrees of freedom that are used for the computation.

2 The value of the $IDEGFD^{th}$ degree of freedom of $\mathbf{u}_1$ at nodal point IPOINT is stored in RINVEC(1).

3 The minimum and maximum values of the real array $\mathbf{u}_1$ are stored in RINVEC, depending on the value of IINVEC(5).
If IINVEC(5) = 0 (default value), the minimum value is stored in RINVEC(1) and the maximum value in RINVEC(2).
If IINVEC(5) = 1, not only the minimum value and maximum value are stored in RINVEC(1) respectively RINVEC(2), but also the co-ordinates of the point for which these values are reached are stored in RINVEC. At output RINVEC(3) - RINVEC(5) contain the co-ordinates for which the minimum is reached and RINVEC(6) - RINVEC(8) contain the co-ordinates for which the maximum is reached.
IINVEC(5) = 2 has the same meaning as IINVEC(5) = 1, however, instead of $\mathbf{u}_1(i)$,
$\mid \mathbf{u}_1(i) \mid$ is used, i.e. the minimum and maximum absolute values of the real array $\mathbf{u}_1$ are computed.
Restriction: if IINVEC(5) > 0 only one unknown per point may be considered.

4 The inner product of $\mathbf{u}_1$ and $\mathbf{u}_2$ is stored in RINVEC(1). When $\mathbf{u}_1$ and $\mathbf{u}_2$ are complex vectors, the inner product is complex too. If one degree of freedom must be used (IDEGFD>0), but essential boundary conditions must be excluded, IINVEC(5) must be made equal to -1 instead of the default value 0.
The combination one of the vectors real and the other one complex is not yet allowed.
At output IINVEC(5) is equal to the total number of degrees of freedom that is used for the computation.

5 The norm of $\mathbf{u}_1$, or the difference of two vectors $\mathbf{u}_1 - \mathbf{u}_2$ is computed and stored in RINVEC(1).
If IINVEC(5) = 0 (default value) $\parallel \mathbf{u}_1 \parallel$ is computed, if IINVEC(5) = 1 $\parallel \mathbf{u}_1 - \mathbf{u}_2 \parallel$ is computed.
The type of norm to be evaluated must be stored in IINVEC(6) (ICHNRM). Possible values:

1. $\parallel \mathbf{u} \parallel = \sum_{i=1}^{N} \mid u_i \mid$
2. $\parallel \mathbf{u} \parallel = \left(\sum_{i=1}^{N} u_i^2 \right)^{\frac{1}{2}}$
3. $\parallel \mathbf{u} \parallel = \max_{1 \le i \le N} \mid u_i \mid$

4. $\| \mathbf{u} \| = \frac{\sum_{i=1}^{N} |u_i|}{N}$
5. $\| \mathbf{u} \| = \sqrt{\sum_{i=1}^{N} u_i^2 / N}$
6. $\| \mathbf{u} \| = \int_{\Omega} | u | \, d\Omega$
7. $\| \mathbf{u} \| = \left(\int_{\Omega} | u |^2 \, d\Omega \right)^{\frac{1}{2}}$
8. $\| \mathbf{u} \| = \frac{\int_{\Omega} |u| d\Omega}{\int_{\Omega} d\Omega}$
9. $\| \mathbf{u} \| = \frac{\left(\int_{\Omega} |u|^2 d\Omega \right)^{\frac{1}{2}}}{\int_{\Omega} d\Omega}$
10. $\| \mathbf{u} \| = \max_{1 \le i \le N} | u_i |$

    The nodal point number, where the maximum value is reached is stored in IINVEC(4)

N is defined as follows:
When IDEGFD=0, N denotes the number of degrees of freedom. When IDEGFD>0, N denotes the number of nodal points where $\mathbf{u}$ is defined, u_i is the i^{th} component of $\mathbf{u}$, and u is the function that is represented by the $JDEGFD^{th}$ degree of freedom of $\mathbf{u}$.
The possibilities 6-9, as well as IDEGFD=-1 are not yet available.

26 A constant is subtracted from a vector: $\mathbf{u}_{out}$ = $\mathbf{u}_1$ - RINVEC(1).
When $\mathbf{u}_1$ is a complex vector, $\mathbf{u}_{out}$ is also a complex vector, and RINVEC(1) must be a complex quantity. INVEC1 and IRESVC may be the same arrays in which case the result is stored in $\mathbf{u}_1$.

27 Linear combination of two vectors: $\mathbf{u}_{out}$ = RINVEC(1) $*\mathbf{u}_1$ + RINVEC(2) $*\mathbf{u}_2$.
When $\mathbf{u}_1$ and $\mathbf{u}_2$ are complex vectors, then $\mathbf{u}_{out}$ is also a complex vector, and RINVEC(1)-(2) must be complex quantities. INVEC1 and IRESVC may be the same arrays in which case the result is stored in $\mathbf{u}_1$.
The combination one of the vectors real and the other one complex is not yet allowed.

28 The value of the $IDEGFD^{th}$ degree of freedom of $\mathbf{u}_1$ at nodal point IPOINT is made equal to RINVEC(1). When $\mathbf{u}_1$ is a complex vector, RINVEC(1) must be a complex quantity.

29 The modulus of the complex $\mathbf{u}_1$, multiplied by the real constant RINVEC(1) is stored in the real vector $\mathbf{u}_{out}$. When RINVEC(1) = 0 is submitted, no multiplication takes place.
$\mathbf{u}_{out}$ = RINVEC(1) (Re($\mathbf{u}_1^2$) + Im ($\mathbf{u}_1^2$))$^{\frac{1}{2}}$

30 The phase of the complex $\mathbf{u}_1$, multiplied by the real constant RINVEC(1) is stored in the real vector $\mathbf{u}_{out}$. When RINVEC(1) = 0 is submitted, no multiplication takes place.
$\mathbf{u}_{out}$ = ATAN2 (Im($\mathbf{u}_1$) , Re($\mathbf{u}_1$)), with ATAN2 the standard FORTRAN 77 function.
(- $\pi < \mathbf{u}_{out}(i) \le \pi$).
If ISPEC (IINVEC(5)) is equal to zero (default) the result is computed in radians, if ISPEC=1, it is transformed to degrees by premultiplication by $180/\pi$.
Remark: When $\mathbf{u}_1$(i) = (0,0) the phase is made equal to 0.

31 User defined function of two SEPRAN vectors: $\mathbf{u}_{out} = \mathbf{f}(\mathbf{u}_1, \mathbf{u}_2)$. The user must define the function through the user subroutine FUNALG (if $\mathbf{u}_1$ and $\mathbf{u}_2$ are real) or FUNALC if both are complex. The combination real/complex is not allowed.
For a description of the subroutines FUNALG and FUNALC see the Users Manual Section 3.3.1.
Restriction: only one unknown per point may be considered.

32 User defined function of n SEPRAN vectors and the co-ordinates:
$\mathbf{u}_{out} = \mathbf{f}(\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n, \mathbf{x})$. The user must define the function through the user subroutine FUNVEC (12.3). The number of vectors n must be stored in IINVEC(5). (Default value: 1).
The n vectors itself must be referenced through the parameter INVEC1, which must be a two-dimensional array: INVEC1(1:5,1:n).
Restriction: At this moment possibility 32 is restricted to the case that $\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n$ all have the same structure. Only one unknown per point may be considered.
The parameters IINVEC and RINVEC are passed undisturbed to the subroutine FUNVEC, hence the user may utilize the free positions to transport information from the main program to subroutine FUNVEC.

33 The length of a vector is stored component-wise into $\mathbf{u}_{out}$, i.e. for two-dimensional vectors $u_{out}(i) = (|u_1(i)|^2 + |u_2(i)|^2)^{\frac{1}{2}}$. and for three-dimensional vectors
$u_{out}(i) = (|u_1(i)|^2 + |u_2(i)|^2 + |u_3(i)|^2)^{\frac{1}{2}}$.
At this moment it is supposed that the components u_1, u_2 and u_3 are all components of one vector $\mathbf{u}_1$. The dimension (2 or 3) must be stored into IINVEC(5) (default value: NDIM = KMESH(6)) and the components are referenced through the parameters IDEGFD1, IDEGFD2 and IDEGFD3, which must have been stored in IINVEC(8) - (10). Default values 1 to 3. IDEGFDi defines the i^{th} degree of freedom in each node.
The output vector is a vector with exactly one degree of freedom in each node. As a consequence it is necessary that either the solution vector has this structure or that there is a type of vector of special structure with exactly one degree of freedom per point.
The output vector is always real, even if the input vector is complex.
IINVEC(3) is not used, i.e. there is no difference between internal unknowns and boundary values.

34 The complex conjugate of the complex $\mathbf{u}_1$ is stored in the complex vector $\mathbf{u}_{out}$.

35 User defined function of two SEPRAN vectors: $\mathbf{u}_{out} = \mathbf{f}(\mathbf{u}_1, \mathbf{u}_2)$. The user must define the function through the user subroutine FUNADA. Both vectors must be real. This option is especially meant for adaptive mesh refinements.

36 Vector consisting of exactly one of the degrees of freedom from the input vector.
The $IDEGFD^th$ degree of freedom of the vector $\mathbf{u}_1$, multiplied by the real constant RINVEC(1) is stored in the vector $\mathbf{u}_{out}$. When RINVEC(1) = 0 is submitted, no multiplication takes place.
IDEGFD must be stored in IINVEC(3).

37 The real part of the complex $\mathbf{u}_1$, multiplied by the real constant RINVEC(1) is stored in the real vector $\mathbf{u}_{out}$. When RINVEC(1) = 0 is submitted, no multiplication takes place.
$\mathbf{u}_{out}$ = RINVEC(1) Re($\mathbf{u}_1^2$)

38 The imaginary part of the complex $\mathbf{u}_1$, multiplied by the real constant RINVEC(1) is stored in the real vector $\mathbf{u}_{out}$. When RINVEC(1) = 0 is submitted, no multiplication takes place.
$\mathbf{u}_{out}$ = RINVEC(1) Im($\mathbf{u}_1^2$)

39 Linear combination of two vectors: $\mathbf{u}_{out}$ = RINVEC(1) $*\mathbf{u}_1$ + RINVEC(2) $*\mathbf{u}_2$.
As extension of 27 the output vector is set equal to RINVEC(3) at those places where otherwise $\mathbf{u}_{out} \leq$ RINVEC(3).

40 Component-wise product of two vectors: $\mathbf{u}_{out}(i) =$ RINVEC(1) $\mathbf{u}_1(i)\mathbf{u}_2(i)$.

41 Point-wise inner-product of two vectors: $\mathbf{u}_{out}(i) = \sum_j \mathbf{u}_1(i)_j + \mathbf{u}_2(i)_j$.
If IDEGFD=0 the summation is carried out over all components in the nodes, if IDEGFD>0, only over the first IDEGFD components.

42 Limit u to values between rinvec(1) and rinvec(2)

43 uout is redefined function of vec1.
$\mathbf{u}_{out} = \mathbf{f}(\mathbf{u}_1)$.
The type of function is defined by ISPEC.
Possible values:
1. function is a logarithm with base 10.

44 Point-wise uout is defined by $u_{out} = \mathbf{a} \cdot \mathbf{u}_1$, with $\mathbf{a}$ a given vector stored in RINVEC(1-3).
The structure of the output vector must be stored in ISPEC.

45 Point-wise uout is defined by $\mathbf{u}_{out} = u_1\ \mathbf{a}$, with $\mathbf{a}$ a given vector stored in RINVEC(1-3).
The structure of the output vector must be stored in ISPEC.

46 Point-wise uout is defined by $\mathbf{u}_{out} = \mathbf{u}_1 - (\mathbf{u}_1 \cdot \mathbf{n})\ \mathbf{n}$, with $\mathbf{n}$ the outer normal on the curve or surface defined in ISPEC.
Mark that u must have NDIM components per point.

51 $\mathbf{u}_{out}$ is a map of the SEPRAN vector $\mathbf{u}_1$, i.e. the SEPRAN vector $\mathbf{u}_1$ is transformed to a new structure defined by the parameters IPROB and ISPEC(IVEC). If ISPEC=0 the output vector is of type solution vector, otherwise it is a vector of special structure with sequence number ISPEC.
If ISPEC<0, the output vector is a vector of special structure defined per element (116), with -ISPEC degrees of freedom per element. All degrees of freedom are computed by averaging in each element.
If the number of degrees of freedom in the output vector differs from the input vector only the common degrees of freedom are copied.

52 $\mathbf{u}_{out}$ is a map of the SEPRAN vector $\mathbf{u}_1$, i.e. the SEPRAN vector $\mathbf{u}_1$ is transformed to a new structure. In this case it is assumed that the input vector is defined per node and has a constant number of degrees of freedom in each node.
The output vector is defined per element and has the same number of degrees of freedom but now per element. The corresponding problem number iprob is copied from the input vector. The values in line elements and connection elements are made equal to 0.
The values in the other elements are computed by taking the mean value of the values in the nodes.

Remarks

1. IDEGFD=-1, IDEGFD=0 and IDEGFD>0 can be used for vectors of the type of the solution vector and for vectors of special structure. For vectors that are defined per element only IDEGFD=0 is available at this moment.
2. If the result of MANVEC is an output vector (ICHOIS > 25) and IDEGFD $\neq 0$ is used, then the resulting output vector has exactly the same number of degrees as the input vector. The degrees of freedom that are not referred to by IDEGFD are either not changed (already existing array) or set equal to zero (new array).
3. In previous versions of SEPRAN instead of MANVEC the subroutines ALGEBR, VECMAN, ANORM, DIFFVC and MANIVC are used. These subroutines still may be used, but their function is completely replaced by subroutine MANVEC. For a description of these subroutines, see Section 12.4. The functions of these subroutines are not changed and they will be supported in the future, so there is no need to change your old programs.

12.2.2 subroutine MANVECBF

Description

Exactly the same as subroutine MANVEC, however, with two extra parameters IBUFFR and BUFFER.

Heading

```
subroutine manvecbf ( ibuffr, buffer, iinvec, rinvec, invec1,
                      invec2, iresvc, kmesh, kprob )
```

Parameters

INTEGER IBUFFR(∗), IINVEC(∗), INVEC1(5,∗), INVEC2(5), IRESVC(5), KMESH(∗), KPROB(∗)

DOUBLE PRECISION BUFFER(∗), RINVEC(∗)

IINVEC, INVEC1, INVEC2, IRESVC, KMESH, KPROB, RINVEC See MANVEC.

IBUFFR Standard integer buffer array as stored in blank common. The reason to put this parameter in the parameter list is that in this way blank common may be avoided.

BUFFER Standard double precision buffer array. In fact this is exactly the same array as IBUFFR.

12.3 Some user function subroutines called by MANVEC

In Section 12.2 subroutine MANVEC has been treated which may be used for manipulation of SEPRAN vectors. In some case MANVEC requires user supplied function subroutines. This may be subroutines from the following set:

FUNALG See Users manual Section 3.3.1

FUNALC See Users manual Section 3.3.1

FUNVEC (12.3.1)

FUNADA (12.3.2)

In this section we shall treat the subroutines FUNVEC and FUNADA.

12.3.1 subroutine FUNVEC

Description

Subroutine FUNVEC is a user written subroutines that must be provided in some cases if subroutine subroutine MANVEC is called.
See the description of MANVEC for a precise description in which cases FUNVEC should be called.

Call

```
CALL FUNVEC ( RINVEC, REAVEC, NVEC, COOR, OUTVEC )
```

Parameters

INTEGER NVEC

DOUBLE PRECISION RINVEC(∗), REAVEC(∗), COOR(∗), OUTVEC

RINVEC Double precision input array for subroutine MANVEC. This array is passed undisturbed to FUNVEC.

REAVEC Double precision (or complex) vector of length NVEC in which the NVEC vectors are stored for one specific node. So REAVEC(i) contains the value of the i^{th} vector in a node.

NVEC Number of vectors in REAVEC.

COOR Array of length NDIM in which the co-ordinates of the node are stored.

OUTVEC The function to be computed must be stored in OUTVEC by the user.

Input

Array RINVEC has been filled by the user before the call of subroutine MANVEC.
The arrays REAVEC and COOR have been filled by subroutine MANVEC.
NVEC has been given a value by subroutine MANVEC.

Output

The parameter OUTVEC must have been given a value by the user.

Remark

Subroutine FUNVEC is called for each node in the mesh.

12.3.2 subroutine FUNADA

Description

Subroutine FUNADA is a special user written subroutines that must be provided if subroutine subroutine MANVEC is called with ICHOICE=35. This possibility is especially meant for adaptive computations.

Call

```
CALL FUNADA ( VEC1, VEC2, VEC3 )
```

Parameters

DOUBLE PRECISION VEC1, VEC2, VEC3

VEC1 Double precision variable corresponding to array INVEC1 in one point.

VEC2 Double precision variable corresponding to array INVEC2 in the same point.

VEC3 In this parameter the result of the manipulation for this point must be stored.

Input

VEC1 and VEC2 have been given a value by subroutine MANVEC.

Output

The parameter VEC3 must have been given a value by the user.

Remark

Subroutine FUNADA is called for each node in the mesh.

12.4 Old subroutines for vector manipulation

In Section 12.2 subroutine MANVEC has been treated which may be used for manipulation of SEPRAN vectors. In previous versions of SEPRAN some other subroutines have been developed, which take care of some of the tasks that may be carried out by MANVEC. Although not longer recommended, these subroutines still exist and they will be available in future versions of SEPRAN. In this section we will describe these "old" subroutines.

ALGEBR (12.4.1) Perform algebraic manipulations on SEPRAN vectors. Identical to MANVEC, however with less possibilities.

VECMAN (12.4.2) Perform algebraic manipulations on SEPRAN vectors. Identical to MANVEC, however with less possibilities.

ANORM (12.4.3) Compute the norm of a vector or the difference of two vectors.

DIFFVC (12.4.4) Compute the difference between two SEPRAN solution vectors or vectors of special structure.

MANIVC (12.4.5) Compute the componentwise Euclidian norm of a vector and store in a vector.

12.4.1 Subroutine ALGEBR

Description

Algebraic manipulations with one or two standard SEPRAN arrays.

Call

```
CALL ALGEBR ( ICHOICE, IDEGFD, IVEC1, IVEC2, IVEC3, KMESH, KPROB,
              A, B, P, Q, IPOINT )
```

Parameters

INTEGER ICHOICE, IDEGFD, IVEC1(5), IVEC2(5), IVEC3(5), KMESH(∗), KPROB(∗), IPOINT

DOUBLE PRECISION A, B, P, Q

ICHOICE Choice parameter, indicates the type of algebraic manipulation that has to be carried out. Possibilities:

1 A constant is subtracted from a vector.
When VECTOR1 is a real vector, then VECTOR3 = VECTOR1 - A.
When VECTOR1 is a complex vector, then VECTOR3 = VECTOR1 - (A , B).

2 The average value of a vector is computed.
When VECTOR1 is a real vector, then A = average (VECTOR1).
When VECTOR1 is a complex vector, then (A , B) = average (VECTOR1).
At output IPOINT is equal to the total number of degrees of freedom that are used for the computation.

3 Linear combination of two vectors.
When VECTOR1 and VECTOR2 are real vectors, then
VECTOR3 = A × VECTOR1 + B × VECTOR2.
When VECTOR1 and VECTOR2 are complex vectors, then
VECTOR3 = (A , B) × VECTOR1 + (P , Q) × VECTOR2

4 The value of the $IDEGFD^{th}$ degree of freedom of VECTOR1 at nodal point IPOINT is stored in A when VECTOR1 is real, and in (A , B) when VECTOR1 is complex.

5 The value of the $IDEGFD^{th}$ degree of freedom of VECTOR1 at nodal point IPOINT is made equal to A when VECTOR1 is real, and to (A , B) when VECTOR1 is complex.

Warning: In the case of periodical boundary conditions, each unknown at periodical boundaries is coupled to two nodes. This means that a change in the value in one of these points, also effects this value in the other point.

6 The minimum value of VECTOR1 is stored in A, the maximum value in B. VECTOR1 must be a real array.

7 The inner product of VECTOR1 and VECTOR2 is stored in A when VECTOR1 and VECTOR2 are real arrays, and in (A , B) when VECTOR1 and VECTOR2 are complex. At output IPOINT is equal to the total number of degrees of freedom that is used for the computation.

8 The modulus of the complex VECTOR1, multiplied by A is stored in the real vector VECTOR3. When A = 0 is submitted, no multiplication takes place.
VECTOR3 = A ($Re(VECTOR1)^2 + Im(VECTOR1)^2$)$^{\frac{1}{2}}$

9 The phase of the complex VECTOR1 is stored in the real vector VECTOR3.
VECTOR3 = ATAN2 (Im(VECTOR1) , Re(VECTOR1)), with ATAN2 the standard FORTRAN 77 function.
(- $\pi < VECTOR3(i) \leq \pi$).
Remark: When VECTOR1(i) = (0,0) the phase is made equal to 0.

10 The inner product of VECTOR1 and VECTOR2 is stored in A when VECTOR1 and VECTOR2 are real arrays, and in (A , B) when VECTOR1 and VECTOR2 are complex. At output IPOINT is equal to the total number of degrees of freedom that are used for the computation. The difference with ICHOICE = 7 is that degrees of freedom that are prescribed by essential boundary conditions are skipped and periodical boundary conditions are counted for only once.

13 Function of two vectors, i.e. VECTOR3 is a function of VECTOR1 and VECTOR2. The user must define the function through the user subroutine FUNALG (Users Manual 6.5.2) (if VECTOR1 and VECTOR2 are real) or FUNALC (Users Manual 6.5.2) if both are complex. The combination real complex is not allowed.

IDEGFD Depending on the value of IDEGFD, the algebraic manipulations will be performed with respect to all degrees of freedom of the vectors (IDEGFD = 0), or with the $IDEGFD^{th}$ degree of freedom only (IDEGFD > 0).

IVEC1 Integer SEPRAN array, where the information of VECTOR1 is stored. In VECTOR1 only the $IDEGFD^{th}$ degree of freedom in the nodal points is considered when IDEGFD > 0.

IVEC2 Integer SEPRAN array, where the information of VECTOR2 is stored. In VECTOR2 only the $IDEGFD^{th}$ degree of freedom in the nodal points is considered when IDEGFD > 0. IVEC2 is only used when ICHOICE = 3 or 7.
When IVEC2 is used, it must have exactly the same structure as IVEC1. The corresponding arrays must be both real or both complex.

IVEC3 Integer SEPRAN array, where the information of VECTOR3 is stored. In VECTOR3 only the $IDEGFD^{th}$ degree of freedom in the nodal points gets a value when IDEGFD > 0. The other degrees of freedom are copied from VECTOR1. IVEC3 is only used when ICHOICE = 1, 3, 8 or 9.

KMESH See subroutine MANVEC (12.2)

KPROB See subroutine MANVEC (12.2)

A, B, P, Q Real variables, see ICHOICE.
When the vectors to be considered are complex, then (A , B) and (P, Q) form complex pairs.

IPOINT Integer parameter. Is only used when ICHOICE = 2, 4, 5 or 7.

Input

ICHOICE, IDEGFD and A, B, P, Q (when ICHOICE = 1, 3, 5) must have a value. When ICHOICE = 8, A must have a value.
The arrays KMESH, KPROB, IVEC1, and when ICHOICE = 3, 7, IVEC2 must have been filled.

Output

When ICHOICE = 1, 3, 8, 9 then array IVEC3 has been filled.
A (and B when the vectors are complex) have got a value when ICHOICE = 2, 4, 6, 7.

Remarks

1. IDEGFD=0 and IDEGFD>0 can be used for vectors of the type of the solution vector and for vectors of special structure. For vectors that are defined per element IDEGFD=0 is available only at this moment.
2. If the result of ALGEBR is an output vector (ICHOICE = 1, 3, 8 or 9) and IDEGFD≠0 is used, then the resulting output vector has exactly the same number of degrees as the input vector. The degrees of freedom that are not referred to by IDEGFD are either not changed (already existing array) or set equal to zero (new array).

12.4.2 Subroutine VECMAN

Description

Algebraic manipulations with one, two or three standard SEPRAN arrays.

Call

```
CALL VECMAN ( ICHOICE, IDEGFD, IVEC1, IVEC2, IVEC3, KMESH, KPROB, CONST, ICONST )
```

Parameters

INTEGER ICHOICE, IDEGFD, IVEC1(5), IVEC2(5), IVEC3(5), KMESH(∗), KPROB(∗), ICONST(∗)

DOUBLE PRECISION CONST(∗)

ICHOICE Choice parameter, indicates the type of algebraic manipulation that has to be carried out. Possibilities:

1 A constant is subtracted from a vector: VECTOR3 = VECTOR1 - CONST(1).
When VECTOR1 is a complex vector, then VECTOR3 is also a complex vector, and CONST(1) must be either complex, or the complex constant must be stored in CONST(1) and CONST(2) (Real resp. imaginary part).

2 The average value of a vector is computed and stored in CONST(1).
When VECTOR1 is a complex vector and CONST is a real array, then the real part of the constant is stored in CONST(1) and the imaginary part in CONST(2).
At output ICONST(1) is equal to the total number of degrees of freedom that are used for the computation.

3 Linear combination of two vectors: VECTOR3 = CONST(1) × VECTOR1 + CONST(2) × VECTOR2.
When VECTOR1 and VECTOR2 are complex vectors, then VECTOR3 is also a complex vector, and CONST must be either complex, or the first complex constant must be stored in CONST(1) and CONST(2) (Real resp. imaginary part). In the same way the second constant must be stored in CONST(3) and CONST(4) if CONST is real.
The combination one of the vectors real and the other one complex is not yet allowed.

4 The value of the $IDEGFD^{th}$ degree of freedom of VECTOR1 at nodal point ICONST(1) is stored in CONST(1). When VECTOR1 is a complex vector and CONST is a real array, then the real part of the constant is stored in CONST(1) and the imaginary part in CONST(2).

5 The value of the $IDEGFD^{th}$ degree of freedom of VECTOR1 at nodal point ICONST(1) is made equal to CONST(1). When VECTOR1 is a complex vector then CONST(1) must be either complex, or the complex constant must be stored in CONST(1) and CONST(2) (Real resp. imaginary part).

Warning: In the case of periodical boundary conditions, each unknown at periodical boundaries is coupled to two nodes. This means that a change in the value in one of these points, also effects this value in the other point.

6 The minimum value of VECTOR1 is stored in CONST(1), the maximum value in CONST(2). VECTOR1 must be a real array.

7 The inner product of VECTOR1 and VECTOR2 is stored in CONST(1). When VECTOR1 and VECTOR2 are complex vectors and CONST is a real array, then the real part of the inner product is stored in CONST(1) and the imaginary part in CONST(2).
The combination one of the vectors real and the other one complex is not yet allowed.
At output ICONST(1) is equal to the total number of degrees of freedom that is used for the computation.

8 The modulus of the complex VECTOR1, multiplied by the real constant CONST(1) is stored in the real vector VECTOR3. When CONST(1) = 0 is submitted, no multiplication takes place.
VECTOR3 = CONST(1) ($Re(VECTOR1)^2 + Im(VECTOR1)^2$)^{1over2}

9 The phase of the complex VECTOR1 is stored in the real vector VECTOR3.
VECTOR3 = ATAN2 (Im(VECTOR1) , Re(VECTOR1)), with ATAN2 the standard FORTRAN 77 function.
(- $\pi < VECTOR3(i) \leq \pi$).
Remark: When VECTOR1(i) = (0,0) the phase is made equal to 0.

10 The inner product of VECTOR1 and VECTOR2 is stored in CONST(1). When VECTOR1 and VECTOR2 are complex vectors and CONST is a real array, then the real part of the inner product is stored in CONST(1) and the imaginary part in CONST(2). The combination one of the vectors real and the other one complex is not yet allowed.
At output ICONST(1) is equal to the total number of degrees of freedom that are used for the computation. The difference with ICHOICE = 7 is that degrees of freedom that are prescribed by essential boundary conditions are skipped and periodical boundary conditions are counted for only once.

11 The minimum and maximum values of the real array VECTOR1 are computed. Besides that, the co-ordinates for which these values are reached are stored in CONST.
Restriction: only one unknown per point may be considered.
At output CONST(1) contains the minimum, CONST(2) the maximum value, CONST(3) - CONST(5) contain the co-ordinates for which the minimum is reached and CONST(6) - CONST(8) contain the co-ordinates for which the maximum is reached.

12 The minimum and maximum absolute values of the real array VECTOR1 are computed. Besides that, the co-ordinates for which these values are reached are stored in CONST.
Restriction: only one unknown per point may be considered.
At output CONST(1) contains the minimum, CONST(2) the maximum value, CONST(3) - CONST(5) contain the co-ordinates for which the minimum is reached and CONST(6) - CONST(8) contain the co-ordinates for which the maximum is reached.

13 Function of two vectors, i.e. VECTOR3 is a function of VECTOR1 and VECTOR2.
The user must define the function through the user subroutine FUNALG (Users Manual 6.5.2) (if VECTOR1 and VECTOR2 are real) or FUNALC (Users Manual 6.5.2) if both are complex. The combination real/complex is not allowed.

IDEGFD Depending on the value of IDEGFD, the algebraic manipulations will be performed with respect to all degrees of freedom of the vectors (IDEGFD = 0), with all internal degrees of freedom i.e. not with the essential boundary conditions (IDEGFD = -1), or with the $IDEGFD^{th}$ degree of freedom only (IDEGFD > 0).

IVEC1 Integer SEPRAN array, where the information of VECTOR1 is stored. In VECTOR1 only the $IDEGFD^{th}$ degree of freedom in the nodal points is considered when IDEGFD > 0.

IVEC2 Integer SEPRAN array, where the information of VECTOR2 is stored. In VECTOR2 only the $IDEGFD^{th}$ degree of freedom in the nodal points is considered when IDEGFD > 0. IVEC2 is only used when ICHOICE = 3 or 7.
When IVEC2 is used, it must have exactly the same structure as IVEC1. The corresponding arrays must be both real or both complex.

IVEC3 Integer SEPRAN array, where the information of VECTOR3 is stored. In VECTOR3 only the $IDEGFD^{th}$ degree of freedom in the nodal points gets a value when IDEGFD > 0. The other degrees of freedom are copied from VECTOR1. IVEC3 is only used when ICHOICE = 1, 3, 8 or 9.

KMESH See subroutine MANVEC (12.2)

KPROB See subroutine MANVEC (12.2)

CONST Real array for input and output purposes. For the use of CONST see ICHOICE.

ICONST Integer array for input and output purposes. For the use of ICONST see ICHOICE.

Input

ICHOICE and IDEGFD must have a value.
When ICHOICE = 4 or 5, ICONST(1) must have been filled.
When ICHOICE = 1, 3, 5 or 8, array CONST must have been filled.
The arrays KMESH, KPROB, IVEC1, and when ICHOICE = 3, 7, 10 or 13 IVEC2 must have been filled.

Output

When ICHOICE = 1, 3, 8, 9, 10 or 13 array IVEC3 has been filled.
When ICHOICE = 2, 4, 6, 7, 10, 11, 12 or 13 array CONST has been filled.

Remarks

1 IDEGFD=-1, IDEGFD=0 and IDEGFD>0 can be used for vectors of the type of the solution vector and for vectors of special structure. For vectors that are defined per element only IDEGFD=0 is available at this moment.

2 If the result of VECMAN is an output vector (ICHOICE = 1, 3, 8, 9 or 13) and IDEGFD $\neq 0$ is used, then the resulting output vector has exactly the same number of degrees as the input vector. The degrees of freedom that are not referred to by IDEGFD are either not changed (already existing array) or set equal to zero (new array).

12.4.3 Function subroutine ANORM

Description

The norm of a vector $\mathbf{u}_1$, or the difference of two vectors $\mathbf{u}_1 - \mathbf{u}_2$ is computed.

Call

```
VALUE = ANORM ( ICHOICE, ICHNRM, JDEGFD, KMESH, KPROB, ISOL1, ISOL2, IELHLP )
```

Parameters

INTEGER ICHOICE, ICHNRM, JDEGFD, KMESH(∗), KPROB(∗), ISOL1(5), ISOL2(5), IELHLP(∗)

DOUBLE PRECISION ANORM

ICHOICE Choice parameter. Possibilities:

0 $\| \mathbf{u}_1 - \mathbf{u}_2 \|$ is computed.

1 $\| \mathbf{u}_1 \|$ is computed.

ICHNRM Choice parameter to fix the type of norm to be computed. Possibilities:

1. $\| \mathbf{u} \| = \sum_{i=1}^{N} | u_i |$
2. $\| \mathbf{u} \| = \left(\sum_{i=1}^{N} u_i^2 \right)^{\frac{1}{2}}$
3. $\| \mathbf{u} \| = \max_{1 \le i \le N} | u_i |$
4. $\| \mathbf{u} \| = \frac{\sum_{i=1}^{N} |u_i|}{N}$
5. $\| \mathbf{u} \| = \sqrt{\sum_{i=1}^{N} u_i^2 / N}$
6. $\| \mathbf{u} \| = \int_{\Omega} | u | \, d\Omega$
7. $\| \mathbf{u} \| = \left(\int_{\Omega} | u |^2 \, d\Omega \right)^{\frac{1}{2}}$
8. $\| \mathbf{u} \| = \frac{\int_{\Omega} |u| d\Omega}{\int_{\Omega} d\Omega}$
9. $\| \mathbf{u} \| = \frac{\left(\int_{\Omega} |u|^2 d\Omega \right)^{\frac{1}{2}}}{\int_{\Omega} d\Omega}$
10. $\| \mathbf{u} \| = \max_{1 \le i \le N} | u_i |$

    The nodal point number, where the maximum value is reached is stored in IELHLP(1).

N is defined as follows:
When JDEGFD=0, N denotes the number of degrees of freedom. When JDEGFD>0, N denotes the number of nodal points where $\mathbf{u}$ is defined, u_i is the i^{th} component of $\mathbf{u}$, and u is the function that is represented by the $JDEGFD^{th}$ degree of freedom of $\mathbf{u}$.
The possibilities 6-9 are not available.

JDEGFD When JDEGFD = 0 all degrees of freedom from the arrays $\mathbf{u}_1$ and $\mathbf{u}_2$ are used, otherwise (JDEGFD > 0) only the $JDEGFD^{th}$ degree of freedom in each nodal point is used. When ICHNRM > 5, JDEGFD must be larger than zero.

KMESH See subroutine MANVEC (12.2)

KPROB See subroutine MANVEC (12.2)

ISOL1 In this array information concerning the vector $\mathbf{u}_1$ must be stored.

ISOL2 In this array information concerning the vector $\mathbf{u}_2$ must be stored. ISOL2 is only used when ICHOICE=0.

IELHLP This array is only used when ICHNRM=10.

Input

The parameters ICHOICE, ICHNRM and JDEGFD must have a value.
The arrays KMESH, KPROB, ISOL1 and ISOL2 (when ICHOICE=0) must have been filled.

Output

ANORM has got the value of the norm according to the definition ICHNRM.
When ICHNRM = 10, IELHLP(1) has been filled.

12.4.4 Subroutine DIFFVC

Description

Compute the maximal difference of two SEPRAN vectors.

Call

```
CALL DIFFVC ( ICHOICE, IVCTR1, IVCTR2, KPROB, DIFMAX )
```

Parameters

INTEGER ICHOICE, IVCTR1(5), IVCTR2(5), KPROB(∗)

DOUBLE PRECISION DIFMAX

ICHOICE ICHOICE must be equal to 0. Other values are not yet available.

IVCTR1,IVCTR2 Arrays containing information of the vectors to be computed. The vectors must be of the same type.

KPROB See subroutine MANVEC (12.2)

DIFMAX Maximal difference of vectors, i.e.
DIFMAX = max | U1(i) - U2(i) |
where U1 corresponds to IVCTR1 and U2 to IVCTR2.
When U1 and U2 are complex arrays, DIFMAX is the maximal difference of both components.
Hence DIFMAX = max |real part (U1(i) - U2(i))| ,
|imaginary part (U1(i) -U2(i))|

Input

The arrays IVCTR1, IVCTR2 and KPROB must be filled.
ICHOICE must have the value 0.

Output

DIFMAX has got a value.

Remarks IVCTR1 and IVCTR2 must be of the same standard SEPRAN type.
For a complex vector DIFMAX is the maximum difference of both the real and imaginary part of the vectors.

12.4.5 Subroutine MANIVC

Description

Create an output vector by some algebraic manipulation with the input vector.

Call

```
CALL MANIVC ( ICHOICE, IDGFD1, IDGFD2, IDGFD3, INPVEC, IOUTVC, IVEC, KMESH,
              KPROB, A )
```

Parameters

INTEGER ICHOICE, IDGFD1, IDGFD2, IDGFD3, INPVEC(5), IOUTVC, IVEC, KMESH(∗), KPROB(∗)

DOUBLE PRECISION A

ICHOICE Choice parameter, indicates the type of algebraic manipulation that has to be carried out. Possibilities:

1 The input vector is supposed to be a vector with at least two degrees of freedom in the nodal points where the manipulation must be carried out.
The output vector is created by the relation: $OUTVEC(i) = (u^2 + v^2)^{\frac{1}{2}}$, where u is the $IDGFD1^{th}$ degree of freedom in nodal point i, and v is the $IDGFD2^{th}$ one. Hence the Euclidian norm of the vector is computed.

IDGFD1 The first degree of freedom to be considered of the input vector is the degree of freedom IDGFD1.

IDGFD2 The second degree of freedom to be considered of the input vector is the degree of freedom IDGFD2.

IDGFD3 Not yet used.

INPVEC Integer SEPRAN array, containing information concerning the input vector.

IOUTVC Integer SEPRAN array, containing information concerning the output vector.

IVEC Indication of the type of the output vector. When IVEC = 0, the output vector gets the structure of the solution vector (ISOL), when IVEC > 0, the output vector is stored as an array of special structure with sequence number IVEC.

KMESH See subroutine MANVEC (12.2)

KPROB See subroutine MANVEC (12.2)

A Not used.

Input

ICHOICE, IDGFD1, IDGFD2, and IVEC must have a value.
The arrays KMESH, KPROB and INPVEC must have been filled.

Output

Array IOUTVC has been filled.

12.5 Copying of SEPRAN vectors (subroutines COPYVC and COPYVEC)

At this moment there are six SEPRAN subroutines that may be used to copy a SEPRAN vector into another SEPRAN vector. These subroutines are:

COPYVC (12.5.1) Copy a complete SEPRAN vector into another one.

COPYVCBF (12.5.2) Same as COPYVC, however, with extra parameters IBUFFR and BUFFER.

COPYVEC (12.5.3) Copy a complete SEPRAN vector or a part of it into another one.

COPYVECBF (12.5.4) Same as COPYVEC, however, with extra parameters IBUFFR and BUFFER.

COPYDF (12.5.5) Copy the contents of one degree of freedom in one array into another degree of freedom in another array possibly corresponding to another problem

COPYDFBF (12.5.6) Same as COPYDF, however, with extra parameters IBUFFR and BUFFER.

In fact COPYVEC may be considered as an extension of COPYVC

12.5.1 subroutine COPYVC

Description

Copy an existing standard SEPRAN vector into another new or existing vector.

Heading

```
subroutine copyvc ( iold, inew )
```

Parameters

INTEGER IOLD(5), INEW(5)

IOLD Integer array containing information about the existing SEPRAN vector to be copied. Length: 5 positions.
IOLD may correspond to a solution vector, a vector of special structure, or a vector defined per element.

INEW In this array information concerning the new array is stored. Array INEW is not the same as IOLD, but the array it refers to is an exact copy of the array IOLD refers to.

Input

Array IOLD must have been filled by a SEPRAN subroutine.

Output

If INEW was already filled with a vector of the same type as IOLD, then the old vector is overwritten, otherwise a new array is created.
INEW(i) = IOLD(i); i=2,3,4,5
INEW(1) contains the pointer to the duplicate vector.

12.5.2 subroutine COPYVCBF

Description

Exactly the same as subroutine COPYVC, however, with two extra parameters IBUFFR and BUFFER.

Heading

```
subroutine copyvcbf ( ibuffr, buffer, iold, inew )
```

Parameters

INTEGER IBUFFR(∗), IOLD(5), INEW(5)

DOUBLE PRECISION BUFFER(∗)

IOLD,INEW See COPYVC.

IBUFFR Standard integer buffer array as stored in blank common. The reason to put this parameter in the parameter list is that in this way blank common may be avoided.

BUFFER Standard double precision buffer array. In fact this is exactly the same array as IBUFFR.

12.5.3 Subroutine COPYVEC

Description

Copy an existing standard SEPRAN vector or a part of it into another new or existing vector.

Heading

```
subroutine copyvec ( iold, inew, kprob, iincopy )
```

Parameters

INTEGER IOLD(5), INEW(5), KPROB(∗), IINCOPY(∗)

IOLD Integer array containing information about the existing SEPRAN vector to be copied. Length: 5 positions.
IOLD may correspond to a solution vector, a vector of special structure, or a vector defined per element.

INEW In this array information concerning the new array is stored. Array INEW is not the same as IOLD, but the array it refers to is an exact copy of the array IOLD refers to.

KPROB Standard SEPRAN array containing information about the problem description. KPROB must have been filled before for example by a SEPRAN start routine that reads the mesh.

IINCOPY Integer input array in which the user defines which copying actions should be performed.
The entries of IINCOPY must be filled as follows:

1 LAST Number of entries of IINCOPY that are filled by the user. If IINCOPY(1) is equal to 0 or 1 only defaults are used. For all positions beyond position IINCOPY(1), defaults are used. If a user fills an entry of IINCOPY with a zero, also the default value is used.

2 COPY_PART defines whether the complete array is copied or not. Possible values:

- 0 The complete array is copied.
- 1 Only the part not corresponding to essential boundary conditions is copied.
- 2 Only the part corresponding to essential boundary conditions is copied.
- 3 Special possibility: Copy the coordinates of the mesh as stored in IOLD (= KMESH) into INEW. Hence the coordinates are stored as a solution vector or vector of special structure.

Default value: 0

3 IPROB, problem number of the output vector INEW.
This parameter is only used if copy_part = 3.

Default value: 1

Input

The arrays IOLD and KPROB must have been filled by a SEPRAN subroutine.
Array IINCOPY must have been filled by the user.

Output

If INEW was already filled with a vector of the same type as IOLD, then the old vector is overwritten, otherwise a new array is created.
INEW(i) = IOLD(i); i=2,3,4,5
INEW(1) contains the pointer to the duplicate vector.

12.5.4 subroutine COPYVECBF

Description

Exactly the same as subroutine COPYVEC, however, with two extra parameters IBUFFR and BUFFER.

Heading

```
subroutine copyvecbf ( ibuffr, buffer, iold, inew, kprob, iincopy )
```

Parameters

INTEGER IBUFFR(∗), IOLD(5), INEW(5), KPROB(∗), IINCOPY(∗)

DOUBLE PRECISION BUFFER(∗)

IOLD,INEW,KPROB,IINCOPY See COPYVEC.

IBUFFR Standard integer buffer array as stored in blank common. The reason to put this parameter in the parameter list is that in this way blank common may be avoided.

BUFFER Standard double precision buffer array. In fact this is exactly the same array as IBUFFR.

12.5.5 Subroutine COPYDF

Description

Copy the contents of one degree of freedom in array ivec1 belonging to kprob1 into another degree of freedom in array ivec2 belonging to kprob2

Heading

```
subroutine copydf ( ivec1, ivec2, kprob1, kprob2, idf1, idf2, kmesh )
```

Parameters

INTEGER IVEC1(5), IVEC2(5), KPROB1(∗), KPROB2(∗), IDF1, IDF2, KMESH(∗)

IVEC1 Integer array containing information about the existing SEPRAN vector to be copied. Length: 5 positions.
IVEC1 may correspond to a solution vector, a vector of special structure, or a vector defined per element.
It must correspond to the problem stored in KPROB1.

IVEC2 In this array information concerning the array in which the copying takes place, is stored.
IVEC2 may correspond to a solution vector, a vector of special structure, or a vector defined per element. The array must already have been created.
It must correspond to the problem stored in KPROB2.

KPROB1 Standard SEPRAN array containing information about the problem description. KPROB1 must have been filled before for example by a SEPRAN start routine that reads the mesh.

KPROB2 Standard SEPRAN array containing information about the problem description. KPROB2 must have been filled before for example by a SEPRAN start routine that reads the mesh.
KPROB2 may be the same as KPROB1, but it may also correspond to a different problem on the same mesh.

IDF1 degree of freedom in ivec1 to be copied

IDF2 degree of freedom in ivec2 to be copied

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

Input

The arrays iVEC1, IVEC2, KPROB1, KPROB2 and KMESH must have been filled by a SEPRAN subroutine.
IDF1 and IDF2 must have a value.

Output

Degree of freedom IDF1 from IVEC1 is copied into degree of freedom IDF2 from IVEC2.

Subroutine COPYDFBF

12.5.6 subroutine COPYDFBF

Description

Exactly the same as subroutine COPYDF, however, with two extra parameters IBUFFR and BUFFER.

Heading

```
subroutine copydfbf ( ibuffr, buffer, ivec1, ivec2, kprob1,
                      kprob2, idf1, idf2, kmesh )
```

Parameters

INTEGER IBUFFR(∗), IVEC1(5), IVEC2(5), KPROB1(∗), KPROB2(∗), IDF1, IDF2, KMESH(∗)

DOUBLE PRECISION BUFFER(∗)

IVEC1, IVEC2, KPROB1, KPROB2, IDF1, IDF2, KMESH See COPYDF.

IBUFFR Standard integer buffer array as stored in blank common. The reason to put this parameter in the parameter list is that in this way blank common may be avoided.

BUFFER Standard double precision buffer array. In fact this is exactly the same array as IBUFFR.

12.6 Addition of SEPRAN matrices

In this Section we describe the subroutines that may be used to add SEPRAN matrices in order to create a new matrix. This subroutine is especially used in time-dependent problems.
At this moment we have two subroutines available:

ADDMAT (12.6.1) is used to make a linear combination of two matrices and store the result in the first one.

ADDMT1 (12.6.2) makes a linear combination of three real matrices and stores the result in a complex matrix.

12.6.1 Subroutine ADDMAT

Description

Compute α AMAT1 + β AMAT2 and store the result in AMAT1, where AMAT1 is a large matrix and AMAT2 may be a large matrix, or a diagonal matrix stored as a solution vector, or right-hand-side vector.

Call

```
CALL ADDMAT ( KPROB, MATR1, MATR2, INTMAT, ALPHA1, ALPHA2, BETA1, BETA2 )
```

Parameters

INTEGER KPROB(∗), MATR1(5), MATR2(5), INTMAT(5)

DOUBLE PRECISION ALPHA1, ALPHA2, BETA1, BETA2

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

MATR1 Integer array containing information concerning the large matrix AMAT1.
Output of a SEPRAN matrix building subroutine, for example BUILD (7.2) (7.2)

MATR2 Integer array containing information concerning the large matrix AMAT2.
Output of a SEPRAN matrix building subroutine, for example BUILD (7.2) (7.2)
When AMAT2 is stored as a large matrix, AMAT2 must have exactly the same structure as MATR1 and correspond to the same array INTMAT.
When AMAT2 is a diagonal matrix it may be stored as a solution vector or a right-hand-side vector. AMAT2 must correspond to the same array KPROB as AMAT1.

INTMAT Integer array containing information concerning the storage of the large matrix AMAT1, and if necessary of AMAT2.
Output of a SEPRAN starting subroutine (Section 4.2) or subroutine COMMAT (4.5.2) (4.5)

ALPHA1,ALPHA2 The multiplication factor α must be stored in the reals ALPHA1 and ALPHA2. When MATR1 corresponds to a real matrix α = ALPHA1, otherwise α = ALPHA1 + i ALPHA2.

BETA1,BETA2 The multiplication factor β must be stored in the reals BETA1 and BETA2. When MATR1 corresponds to a real matrix β = BETA1, otherwise β = BETA1 + i BETA2.

Input

ALPHA1, ALPHA2, BETA1 and BETA2 must have a value.
The arrays MATR1, MATR2, INTMAT and KPROB must have been filled.

Output

The large matrix corresponding to MATR1 has been changed.

Remark

When MATR1 corresponds to a real matrix, MATR2 must also correspond to a real matrix.

12.6.2 Subroutine ADDMT1

Description

Compute α AMAT1 + β AMAT2 + γ AMAT3 and store the result in AMAT4, where AMAT1, AMAT2 and AMAT3 are large real matrices, α, β and γ are complex numbers and the resulting matrix AMAT4 is complex.

Call

```
CALL ADDMT1 ( KPROB, MATR1, MATR2, MATR3, MATR4, INTMT1,
              INTMT2, ALPHA, BETA, GAMMA )
```

Parameters

INTEGER KPROB(∗), MATR1(5), MATR2(5), MATR3(5), MATR4(5), INTMT1(5), INTMT2(5)

COMPLEX ∗ **16** ALPHA, BETA, GAMMA

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

MATR1 Integer array containing information concerning the large matrix AMAT1.
Output of a SEPRAN matrix building subroutine, for example BUILD (7.2)

MATR2 Integer array containing information concerning the large matrix AMAT2.
Output of a SEPRAN matrix building subroutine, for example BUILD (7.2)

MATR3 Integer array containing information concerning the large matrix AMAT3.
Output of a SEPRAN matrix building subroutine, for example BUILD (7.2)

MATR4 Integer array containing information concerning the large matrix AMAT4.

INTMT1 Integer array containing information concerning the storage of the large matrices AMAT1, AMAT2 and AMAT3.
Output of a SEPRAN starting subroutine (Section 4.2) or subroutine COMMAT (4.5.2) (4.5)

INTMT2 Integer array containing information concerning the storage of the large matrix AMAT4.
Output of a SEPRAN starting subroutine (Section 4.2) or subroutine COMMAT (4.5.2) (4.5)

ALPHA The multiplication factor α.

BETA The multiplication factor β.

GAMMA The multiplication factor γ.

Input

ALPHA, BETA and GAMMA must have a value.
The arrays MATR1, MATR2, MATR3, INTMT1, INTMT2 and KPROB must have been filled.

Output

MATR4 has been filled.

12.7 Copying of SEPRAN matrices

In this section we treat the subroutines that are available to copy SEPRAN matrices into other SEPRAN matrices. At this moment there is only one such subroutines available: COPYMT.

Description

Copy a SEPRAN matrix into another new or existing matrix. The matrix may be a diagonal matrix stored as right-hand side.

Call

```
CALL COPYMT ( MATROL, MATRNW, KPROB )
```

INTEGER MATROL(5), MATRNW(5), KPROB(∗)

MATROL Integer array containing information about the existing SEPRAN matrix to be copied.

MATRNW In this array information concerning the new matrix is stored. Array MATRNW is not the same as MATROL, but the array it refers to is an exact copy of the array MATROL refers to.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

Input

The arrays KPROB and MATROL must have been filled by a SEPRAN subroutine.

Output

If MATRNW was already filled with a matrix of the same type as MATROL, then the old matrix is overwritten, otherwise a new array is created.

12.8 Matrix vector multiplication (Subroutine MAVER)

Description

Matrix-vector multiplication $\mathbf{Sx} = \mathbf{y}$.
The matrix may not have been changed by a linear solver, for example subroutine SOLVEL (8.3).

Call

```
CALL MAVER ( MATR, IX, IY, INTMAT, KPROB, ICHOICE )
```

INTEGER MATR(5), IX(5), IY(5), INTMAT(5), KPROB(∗), ICHOICE

MATR Output of one of the matrix building subroutines, for example subroutine BUILD (7.2). MATR must contain information concerning the matrix **S**.

IX must be a standard SEPRAN array (for example ISOL), containing information of the vector **x**.

IY is a standard SEPRAN array, which contains information of the vector **y** at output. IY may be an existing array, in which case the corresponding array **y** is changed.

INTMAT Array containing information about the matrix structure. INTMAT must be output of one of the starting subroutines (Section 4.2) or subroutine COMMAT (4.5)

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

ICHOICE Choice parameter indicating whether boundary conditions must be taken into account for the matrix multiplication or not, and also how boundary conditions must be filled in the array **y**.
Possibilities:

0 Boundary conditions are not taken into account in the multiplication.
The positions in **y** corresponding to boundary conditions are left unchanged.

1 Boundary conditions are taken into account in the multiplication.
The positions in **y** corresponding to boundary conditions are left unchanged.

2 Boundary conditions are not taken into account in the multiplication.
The positions in **y** corresponding to boundary conditions get the value of the same positions in **x**.

3 Boundary conditions are taken into account in the multiplication.
The positions in **y** corresponding to boundary conditions get the value of the same positions in **x**.

4 Boundary conditions are not taken into account in the multiplication.
The positions in **y** corresponding to boundary conditions get the value zero.

5 Boundary conditions are taken into account in the multiplication.
The positions in **y** corresponding to boundary conditions get the value zero.

6 The vector **y** is made equal to the effect of the boundary conditions without the matrix vector multiplication. The positions corresponding to the boundary conditions get the value **0**.
Hence $\mathbf{y}_f = \mathbf{S}_{fp}\mathbf{x}_p$ $\mathbf{y}_p = \mathbf{0}$. (See Remark for a definition of $\mathbf{y}_f, \mathbf{S}_{fp}, \mathbf{x}_p$ and $\mathbf{y}_p$).

7 The existing vector y is updated with minus the effect of the boundary conditions. Hence $\mathbf{y}_f = \mathbf{y}_f - \mathbf{S}_{fp}\mathbf{x}_p$. This is typically meant for an update of a right-hand side vector where the effect of the boundary conditions has not been taken into account

8 The existing vector y is filled with the reaction forces that result from the essential boundary conditions. Hence $\mathbf{y}_p = \mathbf{S}_{pf}\mathbf{x}_f + \mathbf{S}_{pp}\mathbf{x}_p$ This option is only available if IBC-MAT=1 (See Users Manual, Section 3.2.4) and is only useful if $\mathbf{x}_f$ has already been computed. The part corresponding to $\mathbf{y}_f$ is left unchanged, which is a fast option if only the reaction forces are required.

9 Combination of 1 and 8.

10 See 8, however, in this only the reaction forces *yp* are computed. The vector is cleared before this computation, hence all other entries of the vector are zero.

Input

ICHOICE must have a value.
The arrays MATR, IX, INTMAT and KPROB must have been filled.

Output

Array IY has been filled.
The result of the matrix vector multiplication has been stored in the corresponding array $\mathbf{y}$.

Remark

Let the free degrees of freedom in $\mathbf{x}$ and $\mathbf{y}$ be denoted by $\mathbf{x}_f$ resp. $\mathbf{y}_f$. Let the prescribed degrees of freedom in $\mathbf{x}$ and $\mathbf{y}$ be denoted by $\mathbf{x}_p$ resp. $\mathbf{y}_p$. Then the matrix $\mathbf{S}$ can be written as:

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_{ff}\mathbf{S}_{fp} \\ \mathbf{S}_{pf}\mathbf{S}_{pp} \end{bmatrix} \tag{12.8.1}$$

When ICHOICE = 0, 2 or 4 then the result of MAVER is: $\mathbf{y}_f = \mathbf{S}_{ff}\mathbf{x}_f$.
When ICHOICE = 1, 3 or 5 then the result of MAVER is: $\mathbf{y}_f = \mathbf{S}_{ff}\mathbf{x}_f + \mathbf{S}_{fp}\mathbf{x}_p$.
When ICHOICE = 0, 1 $\mathbf{y}_p$ is left unchanged, when ICHOICE = 2, 3 $\mathbf{y}_p = \mathbf{x}_p$, otherwise when ICHOICE = 4, 5 $\mathbf{y}_p = 0$.

12.9 Local transformations and back transformations (subroutine LOCTRN)

Description

This subroutine transforms the boundary conditions of a solution vector into the transformed form in case of a local transform or vice-versa, i.e. performs the back-transformation.

Call

```
CALL LOCTRN ( ITRANS, ISOL, KPROB )
```

INTEGER ITRANS, ISOL(5), KPROB(∗)

ITRANS Indicates if a transformation from standard vector (i.e. usual degrees of freedom, for example Cartesian) to transformed vector (i.e. with transformed degrees of freedom in points with local transform, for example normal and tangential components) is defined (1) or that the backwards transformation to standard vector must be computed (2).

ISOL Standard SEPRAN array referring to a SEPRAN solution vector. This is the vector to be transformed or backtransformed.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

Input

ITRANS must have a value.
The arrays ISOL and KPROB must have been filled.

Output

The array corresponding to ISOL has been changed.

12.10 Computation of principal stresses (subroutine PRINCIPSTR)

In this section we describe the subroutines to compute the principal stresses once the stress tensor has been computed.
At this moment we have the following subroutines:

PRINCIPSTR Standard routine to compute the principal stresses using blank common.

PRINCIPSTRBF Exactly the same routine as PRINCIPSTR, but instead of blank common two extra parameters BUFFER and IBUFFR are used.

12.10.1 Subroutine PRINCIPSTR

Description

The principal stresses are computed from an already given stress tensor as described in the users manual Section 3.2.3.

Heading

```
subroutine principstr ( isol, iinput, kmesh )
```

Parameters

INTEGER ISOL(5,∗), IINPUT(10), KMESH(∗)

ISOL SEPRAN array containing information of solution vectors. In this array information of both the given stress tensor as the computed principal stress tensor must be stored.

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

IINPUT integer input array with the following contents.

Pos. 1 iseqstress Sequence number of array stress in isol.
This means that information of the input stress tensor must be stored in ISOL(.,iseqstress)

Pos. 2 iseqeigval Sequence number of array eigval in isol.
If the array of eigenvalues must be computed, the information about this vector is stored in ISOL(.,iseqeigval).

Pos. 3 iseqeigvec Sequence number of array eigvec in isol.
If the principal stress tensor must be computed, the information about this tensor is stored in ISOL(.,iseqeigvec).

Pos. 4 ichoice Integer choice parameter with the following meaning:

1 Compute the eigenvalues only
2 Compute the eigenvectors only
3 Compute the scaled eigenvectors only
4 Compute the eigenvectors and eigenvalues
5 Compute the scaled eigenvectors and eigenvalues

Pos. 5-10 Not yet used.
These parameters must have the value 0.

Input

The arrays KMESH, ISOL and IINPUT must be filled.

Output

The required vector is computed and information stored in ISOL.

12.10.2 Subroutine PRINCIPSTRBF

Description

The principal stresses are computed from an already given stress tensor as described in the users manual Section 3.2.3.

Heading

```
subroutine principstrbf ( ibuffr, buffer, isol, iinput, kmesh )
```

Parameters

INTEGER IBUFFR(∗), ISOL(5,∗), IINPUT(10), KMESH(∗)

DOUBLE PRECISION BUFFER(∗)

ISOL, KMESH, IINPUT See subroutine PRINCIPSTR

IBUFFR Standard integer buffer array as stored in blank common. The reason to put this parameter in the parameter list is that in this way blank common may be avoided.

BUFFER Standard double precision buffer array. I fact this is exactly the same array as IBUFFR.

Input

The arrays KMESH, ISOL and IINPUT must be filled.
The arrays IBUFFR and BUFFER must have been filled.

Output

The required vector is computed and information stored in ISOL.
The arrays IBUFFR and BUFFER have been changed.

12.11 Deleting SEPRAN arrays (subroutine DELSEPAR)

In this section we describe the subroutines to delete SEPRAN arrays hat have been created but are not of use anymore.
At this moment we have the following subroutines:

DELSEPAR Standard routine to delete SEPRAN arrays.

DELSEPABF Exactly the same routine as DELSEPAR, but instead of blank common one extra parameter IBUFFR is used.

12.11.1 Subroutine DELSEPAR

Description

Standard routine to delete SEPRAN arrays.

Heading

```
subroutine delsepar ( iarray )
```

Parameters

INTEGER IARRAY(∗)

IARRAY SEPRAN array that has been created before and must be deleted. IARRAY contains a reference to the real data in IBUFFR.
At this moment only the possibilities IARRAY solution array or matrix has been implemented.
Hence IARRAY may be a vector of type ISOL, IVECTR or MATR.

Input

Array IARRAY must be filled.

Output

The corresponding vector is deleted from the buffer array IBUFFR.

12.11.2 Subroutine DELSEPABF

Description

Standard routine to delete SEPRAN arrays.

Heading

```
subroutine delsepabf ( ibuffr, iarray )
```

Parameters

INTEGER IBUFFR(∗), IARRAY(∗)

IARRAY See subroutine DELSEPAR

IBUFFR Standard integer buffer array as stored in blank common. The reason to put this parameter in the parameter list is that in this way blank common may be avoided.

Input

Array IARRAY must be filled.
Array IBUFFR must have been filled.

Output

The corresponding vector is deleted from the buffer array IBUFFR.

13 Computation of integrals

13.1 Introduction

In this chapter we describe all available subroutines to compute integrals over computed solutions. These integrals may be volume integrals or boundary integrals. With computed solutions we do not only mean solution vectors in the strict sense but also vectors having the structure of a solution vector or vectors of special structure.

The following sections treat the various integration subroutines:

13.2 describes all subroutines that are available for integration over volumes.

13.3 treats all subroutines that are available for integration over boundaries.

13.2 Computation of volume integrals

In this section we treat the subroutines that are available to compute integrals of SEPRAN solution arrays or arrays of special structure over the complete region or over a part of the region.
At this moment the following subroutines are available:

VOLINT (13.2.1) Standard subroutine to compute the integral over a region.

INTEGR (13.2.2) Extension of VOLINT in the sense that extra information about the integration is possible (minimum and maximum values per element). Furthermore the parameter list has been created that future extensions are possible.

INTEGS (13.2.3) Most general of the integration subroutines. INTEGS has exactly the same possibilities, but moreover, it allows input from the standard input file as described in the Users manual Section 3.2.12.

13.2.1 Function subroutine VOLINT

Description

The volume integral over a region or a part of a region is computed, by adding integrals over elements. The possible integrals to be computed for standard problems are described in the manual "Standard Problems". The user may skip element groups for the computation of the integrals. Boundary elements are always skipped.

Call

```
VALUE = VOLINT ( ICHOICE_INT, ICHELI, JDEGFD, KMESH, KPROB, IVECTR,
                 IUSER, USER, IELHLP )
```

INTEGER ICHOICE_INT, ICHELI, JDEGFD, KMESH(∗), KPROB(∗), IVECTR(5), IUSER(∗), IELHLP(∗)

DOUBLE PRECISION USER(∗)

ICHOICE_INT Indicates if the integral must be computed over the complete region or not. Possible values:

- 0 Standard situation; all elements except boundary elements are used for the addition of element integrals.
- 1 Not all element groups are used for the addition of element integrals. See IELHLP.

ICHELI This parameter gives information of the type of integral to be computed. The user must consult the manual "Standard Problems" for the appropriate values of this parameter in the case of type numbers larger than 99.
For user defined problems (type numbers between 1 and 99) the selection of ICHELI is free. See the description of subroutine ELINT as described in the User Manual, Section 4.7.

JDEGFD Only the $JDEGFD^{th}$ degree of freedom in each nodal point is used for the integration.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

IVECTR Array containing information concerning the vector (u) to be integrated (in general the solution vector ISOL).

IUSER,USER These arrays may be used for additional information. Consult the manual "Standard Problems" for the actual filling of these arrays.

IELHLP Help array to be used when ICHOICE_INT = 1.
In IELHLP(IELGRP+5) (1≤IELGRP≤NELGRP), the user must store, whether elements with standard element sequence number IELGRP are added to the integral or not.
When IELHLP(IELGRP+5)=0 the elements with standard element sequence number IELGRP are skipped, when IELHLP(IELGRP+5)=1 these elements are added.
IELHLP is a variable length array, hence position 1 must be filled by the user.

VOLINT At output VOLINT has got the value of the integral.

Input

The arrays KMESH, KPROB, IVECTR and when necessary IUSER and USER must have been filled.
The parameters ICHOICE_INT, ICHELI and JDEGFD must have a value.

Output

VOLINT has got a value.

Subroutines called

Subroutine VOLINT calls element subroutines. These may be standard element subroutines from the library (corresponding to type numbers larger than 99), or user written element subroutines (corresponding to type numbers between 1 and 99).
In Section 4.7 of the Users Manual it is described how user written element subroutines must be programmed.

13.2.2 Subroutine INTEGR

Description

The volume integral over a region or a part of a region is computed, by adding integrals over elements. The possible integrals to be computed for standard problems are described in the manual "Standard Problems". The user may skip element groups for the computation of the integrals. Boundary elements are always skipped.
At the users wish the minimum and maximum values of the integrals per element may be computed.

Call

```
CALL INTEGR ( IINVOL, OUTVAL, KMESH, KPROB, IVECTR, IUSER, USER )
```

INTEGER IINVOL(∗), KMESH(∗), KPROB(∗), IVECTR(5), IUSER(∗)

DOUBLE PRECISION USER(∗), OUTVAL(∗)

IINVOL Integer input array that must be provided by the user.
IINVOL must be filled as follows:

1 Number of entries of IINVOL that are filled by the user.
If IINVOL(1) = 0, only defaults are used

2 ICHELI Choice parameter to indicate which of the possible integrals has to be computed. The value of ICHELI is passed undisturbed to the element subroutines.
Default value: 1

3 JDEGFD Choice parameter to indicate a degree of freedom in a nodal point. The value of JDEGFD is passed undisturbed to the element subroutines.
Default value: 1

4 MAXMIN Indication if only the integral must be computed or also minimum and maximum values of the integrals per element. Possible values:

- 0 Only the integral is computed.
- 1 The integral as well as the minimum and maximum value per element are computed.
- 2 The integral as well as the minimum and maximum absolute value of the integral per element are computed.

Default value: 0

5 ALLELEMENTS Indication if all standard element sequence numbers are used (0) or not (1). In that case the positions 16 to 15+NELGRP+ NUMNATBND (NELGRP is number of element groups and NUMNATBND the number of boundary element groups) are considered as an array IELHLP to be filled by the user in the following way: In IELHLP(IELGRP) (1≤IELGRP≤NELGRP+NUMNATBND) the user must store, whether elements with standard element sequence number IELGRP (or IELGRP = NELGRP + IBNGRP for boundary elements) are added to the large matrix and large vector or not. When IELHLP(IELGRP) = 0 the elements with standard element sequence number IELGRP are skipped, when IELHLP(IELGRP) = 1 these elements are added.
Default value: 0

OUTVAL In this array the computed integral is stored as well as the minimal and maximum values per element if MAXMIN > 0.
The values are stored in the following positions of OUTVAL:

1 integral.

2 minimum value of the integrals per element.

3 maximum value of the integrals per element.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

IVECTR Array containing information concerning the vector (u) to be integrated (in general the solution vector ISOL).

IUSER,USER These arrays may be used for additional information. Consult the manual "Standard Problems" for the actual filling of these arrays.

Input

The arrays IINVOL, KMESH, KPROB, IVECTR and when necessary IUSER and USER must have been filled.

Output

Array OUTVAL has been filled (at least the first position).

Subroutines called

See VOLINT.

13.2.3 Subroutine INTEGS

Description

INTEGS has exactly the same possibilities as INTEGR (13.2.2), but besides that input may be read from the Standard input file as described in the Users Manual Section 3.2.12.

Call

```
CALL INTEGS ( IINVOL, OUTVAL, KMESH, KPROB, IVECTR, IUSER, USER, IREAD )
```

INTEGER IINVOL(∗), KMESH(∗), KPROB(∗), IVECTR(5), IUSER(∗), IREAD

DOUBLE PRECISION USER(∗), OUTVAL(∗)

IINVOL, OUTVAL, KMESH, KPROB, IVECTR, IUSER, USER See INTEGR.

IREAD With this parameter it is indicated if array IINVOL is used or that input is read from the standard INPUT file.
Possible values:

- 0 The input is read from the standard input file.
- >0 The input as read from the standard input file by the SEPRAN starting subroutines (for example SEPSTN (4.2.3. is used. The value of iread defines the sequence number
- -1 No input is read, all information from array IINVOL is used.

Input

The arrays IINVOL, KMESH, KPROB, IVECTR and when necessary IUSER and USER must have been filled.
IREAD must have a value.

Output

Array OUTVAL has been filled (at least the first position).

Subroutines called

See VOLINT (13.2.1).

13.3 Computation of boundary integrals

In this section we treat the subroutines that are available to compute integrals of SEPRAN solution arrays or arrays of special structure along boundaries.
At this moment the following subroutines are available:

BOUNIN (13.3.1) Compute a boundary integral over a part of the boundary. The result is a scalar. BOUNIN is only applicable for R^2.

BOUNIV (13.3.2) Extension of BOUNIN in the sense that more types of integrals may be computed and that the result may also be a vector quantity of length 2.

INTBOU (13.3.3) is an extension of BOUNIV. The boundary may consist of curves in R^2 or surfaces in R^3.

INTBND (13.3.4) is the most general of the subroutines. Compared to INTBOU the subroutine is more flexible in the number of curves or surfaces it allows.
Besides that INTBND allows the user to give the input for the subroutine in the standard input file as described in the Users Manual Section 3.2.14.

13.3.1 Subroutine BOUNIN

Description

Depending on the value of ICHOICE_INT some boundary integral over a part of the boundary is computed.

Call

```
VALUE = BOUNIN ( ICHOICE_INT, IRULE, IDIM, NUMBER, KMESH, KPROB, ICRV1, ICRV2,
                 IVECTR, IUSER, USER )
```

INTEGER ICHOICE_INT, IRULE, IDIM, NUMBER, KMESH(∗), KPROB(∗), ICRV1, ICRV2, IVECTR(∗), IUSER(∗)

DOUBLE PRECISION USER(∗)

ICHOICE_INT Indicates what kind of integral has to be computed. ICHOICE_INT consists of two parts ICHINT and ICHFUN according to the relation
ICHOICE_INT = ICHINT + 100 × ICHFUN.

ICHINT indicates the type of the integral. Possibilities:

1 $\int_{\partial\Omega} f u ds$
2 $\int_{\partial\Omega} \mathbf{u}\cdot\mathbf{n} ds$ with $\mathbf{n}$ the normal.
3 $\int_{\partial\Omega} \mathbf{u}\cdot\mathbf{t} ds$ with $\mathbf{t}$ the tangential vector.
4 $\int_{\partial\Omega} \mathbf{n}\cdot\sigma\cdot\mathbf{n} ds$ with $\mathbf{n}$ the normal and σ the stress tensor.
5 $\int_{\partial\Omega} \mathbf{n}\cdot\sigma\cdot\mathbf{t} ds$ with $\mathbf{n}$ the normal, $\mathbf{t}$ the tangential vector and σ the stress tensor.

ICHFUN Indicates how the function f must be computed. ICHFUN > 0 is only permitted when ICHINT $= 1$. The function f will be computed by the call of the function subroutine FUNC: $f(x, y, z)$ = FUNC (ICHFUN, X, Y, Z).
Function subroutine FUNC must be written by the user. For a description of how to program FUNC, see the Introduction Section 5.5.4.

Remark: The normal is defined as the outward normal if the curve is defined such that it is part of a counter clockwise boundary of the region, otherwise it is the inward normal.
The tangential vector is defined in the direction of the curve.

IRULE The numerical integration to be carried out depends on IRULE. Possibilities:

1 Trapezoid type rule (Integration based on two points).

2 Trapezoid type rule with axi-symmetric co-ordinates, i.e. $ds = 2\pi r ds'$.

3 Simpson type rule (Integration based on three points).

4 Simpson type rule with axi-symmetric co-ordinates, i.e. $ds = 2\pi r ds'$

Remark
When the approximation of the integrand is linear (i.e. two points in each element on the boundary), the trapezoid rule should be used. For higher order approximations (at least three points in each element on the boundary), Simpson's rule may be used.

IDIM Indication on what kind of boundary, the boundary integral must be computed. Possibilities:

1 The boundary integral must be computed over curves. IDIM = 1 may only be used when surfaces or volumes have been created. All curves are considered in the given order. See ICRV1, ICRV2.

-1 See IDIM=1, however, the curves are considered in reversed order.

NUMBER The $NUMBER^{th}$ degree of freedom in each nodal point from the vector corresponding to IVECTR is used. When **u** is a vector (ICHINT = 2 or 3), the degrees of freedom u_1 and u_2 in each nodal point are supposed to be the degrees of freedom NUMBER and NUMBER+1. When σ is a tensor (ICHINT = 4 or 5), the degrees of freedom σ_{11}, σ_{12} and σ_{22} in each nodal point are supposed to be the degrees of freedom NUMBER, NUMBER+1 and NUMBER+2.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

ICRV1,ICRV2 When IDIM = 1, all curves with numbers ICRV1, ICRV1 + 1, . . . , ICRV2 are used in that order. The integral is computed over the curves ICRV1, . . . , ICRV2. When ICRV1 < ICRV2, the order is given by ICRV1, ICRV1+1, . . . , ICRV2.
When IDIM=-1 the curves are used in reversed order, starting with the end point of ICRV1 etc.
When ICHINT=3 the tangential vector is taken into the direction of the curves. When ICHINT=2 the curves must be such that the surface is at the left-hand side of the curves.

IVECTR Array containing information concerning the vector (u) to be integrated (in general the solution vector ISOL).

IUSER,USER User arrays; these are not used with any combination of parameters described in this manual.

BOUNIN At output BOUNIN has got the value of the integral.

Input

The arrays KMESH, KPROB and IVECTR must have been filled.
The parameters ICHOIS_INT, IRULE, IDIM, ICRV1, and ICRV2 must have a value.

Output

BOUNIN has got a value.

Remark

It is necessary that at least in the first and last point of the curves, the $NUMBER^{th}$ degree of freedom is available in the vector corresponding to IVECTR.

13.3.2 Subroutine BOUNIV

Description

BOUNIV performs exactly the same tasks as BOUNIN (13.3.1), however, BOUNIV may also compute the integral over a two-dimensional vector.

Call

```
CALL BOUNIV ( ICHOICE_INT, IRULE, IDIM, NUMBER, KMESH, KPROB, ICRV1, ICRV2,
              IVECTR, IUSER, USER, VALUE )
```

INTEGER ICHOICE_INT, IRULE, IDIM, NUMBER, KMESH(∗), KPROB(∗), ICRV1, ICRV2, IVECTR(∗), IUSER(∗)

DOUBLE PRECISION USER(∗), VALUE(∗)

ICHOICE_INT Indicates what kind of integral has to be computed. ICHOICE_INT consists of two parts ICHINT and ICHFUN according to the relation ICHOICE_INT = ICHINT + 100 × ICHFUN.

ICHINT indicates the type of the integral. Possibilities:

1-5 See BOUNIN (13.3.1).

6 $\int_{\partial\Omega} \mathbf{n} \cdot \sigma ds$, with $\mathbf{n}$ the outward normal, and σ the stress tensor. The result of this integration is a vector.

7 $\int_{\partial\Omega} u\mathbf{n} ds$ with $\mathbf{n}$ the outward normal and u a scalar function. The result of this integration is a vector.

ICHFUN See BOUNIN (13.3.1).

IRULE, IDIM, NUMBER, KMESH, KPROB, ICRV1, ICRV2, IVECTR, IUSER, USER See BOUNIN (13.3.1).

VALUE Array containing the computed integrals at output. When ICHINT = 1 to 5, VALUE has length 1, otherwise value has length 2.

Input

See BOUNIN (13.3.1).

Output

Array VALUE has been filled.

13.3.3 Subroutine INTBOU

Description

INTBOU computes an integral along a boundary. For a two-dimensional region, the boundary is supposed to be a curve For a three-dimensional region the boundary is supposed to be a surface. Other possibilities have not yet been implemented.

Call

```
CALL INTBOU ( IINTBN, RINTBN, KMESH, KPROB, IVECTR, VALUE )
```

INTEGER IINTBN(∗), KMESH(∗), KPROB(∗), IVECTR(∗)

DOUBLE PRECISION RINTBN(∗), VALUE(∗)

IINTBN Integer input array in which the user indicates what type of integral must be computed. IINTBN must be filled as follows:

1 gives the last position of array IINTBN that has been filled by the user. For all other positions only defaults are used.
So: if IINTBN(1)=0 or 1 only defaults are used

2 (ICHINT) indicates what type of integral has to be computed. Possible values:

1 $\int_{\partial\Omega} f u ds$

2 $\int_{\partial\Omega} \mathbf{u}\cdot\mathbf{n} ds$ with $\mathbf{n}$ the normal.

3 $\int_{\partial\Omega} \mathbf{u}\cdot\mathbf{t} ds$ with $\mathbf{t}$ the tangential vector.

4 $\int_{\partial\Omega} \mathbf{n}\cdot\sigma\cdot\mathbf{n} ds$ with $\mathbf{n}$ the normal and σ the stress tensor.

5 $\int_{\partial\Omega} \mathbf{n}\cdot\sigma\cdot\mathbf{t} ds$ with $\mathbf{n}$ the normal, $\mathbf{t}$ the tangential vector and σ the stress tensor.

6 $\int_{\partial\Omega} \mathbf{n}\cdot\sigma ds$, with $\mathbf{n}$ the outward normal, and σ the stress tensor. The result of this integration is a vector.

7 $\int_{\partial\Omega} u\mathbf{n} ds$ with $\mathbf{n}$ the outward normal and u a scalar function. The result of this integration is a vector.

The possibilities 2 to 7 have only been implemented for NDIM = 2, i.e. two-dimensional regions.
Default value: 1

3 (ICHFUN) indication of the value of $f(\mathbf{x})$. Possible values:

0 $f = 1$

>0 f is a function of the space:

```
f = func ( ichfun, x, y, z )       (real case)
       or
f = cfunc ( ichfun, x, y, z )      (complex case)
```

See Section 5.5.4 in the Introduction manual.
At this moment only the combination $f \neq 1$ and ichint=1 is permitted.

Default value: 0

4 (IRULE) The numerical integration rule to be carried out depends on IRULE. Possibilities: Possible values: For curves only (i.e. 2D):

1 trapezoid rule, cartesian co-ordinates

2 trapezoid rule, axisymmetric co-ordinates

3 simpson's rule, cartesian co-ordinates

4 simpson's rule, axisymmetric co-ordinates

For surfaces only (i.e. 3D):

1 Newton cotes rule, i.e. integration rule based on the type of approximation used in the element

Default value: 1

5 (IDIM) This parameter is used for curves only It indicates if the curves are given in the normal order (1) or in reversed order (-1)
Default value: 1

6 (NUMBER) This parameter indicates which unknown (physical degree of freedom) must be used from the (solution) vector u (IVECTR). If ICHINT = 2, 3 or 7, it is supposed that the degrees of freedom are equal to NUMBER, NUMBER+1 and NUMBER+2 (3D). If ICHINT = 4, 5 or 6, it is supposed that the degrees of freedom are equal to NUMBER, NUMBER+1 and NUMBER+2 (2D) or NUMBER to NUMBER+5 (3D).
Default value: 1

7 (ICRV1) This parameter is used in combination with ICRV2.
Default value: 1

8 (ICRV2) This parameter is used in combination with ICRV1.
In 2D the integral is computed over the curves ICRV1 to ICRV2 (IDIM=1) or over these curves in reversed order (IDIM=-1). In the last case the starting point is the last point of ICRV1.
When ICHINT=3 the tangential vector is taken into the direction of the curves. When ICHINT=2, the curve must be such that the surface is at the left- hand side of the curves.
In 3D the integral is computed over the surfaces ICRV1 to ICRV2.
Default value: 1

RINTBN Not yet used.

KMESH, KPROB, IVECTR See BOUNIN (13.3.1).

VALUE Array containing the computed integrals at output. When ICHINT = 1 to 5, VALUE has length 1, otherwise value has length 2.
If the vector to be integrated corresponds to a complex vector, VALUE is a complex array

Input

The arrays KMESH, KPROB, IINTBN and IVECTR must have been filled.

Output

Array VALUE has been filled.

13.3.4 Subroutine INTBND

Description

INTBND computes an integral along a boundary. For a two-dimensional region, the boundary is supposed to be a curve For a three-dimensional region the boundary is supposed to be a surface. The definition of curves and surfaces is more general than in the previous subroutines.
Furthermore, compared to the previous subroutines, INTBND also allows input from the input file.

Call

```
CALL INTBND ( IINTBN, RINTBN, KMESH, KPROB, IVECTR, ICURVS,
              VALUE, IREAD )
```

INTEGER IINTBN(∗), KMESH(∗), KPROB(∗), IVECTR(∗), ICURVS(∗), IREAD

DOUBLE PRECISION RINTBN(∗), VALUE(∗)

IINTBN Integer input array in which the user indicates what type of integral must be computed. IINTBN is only used if IREAD = -1.
IINTBN must be filled as follows:

1 gives the last position of array IINTBN that has been filled by the user. For all other positions only defaults are used.
So: if IINTBN(1)=0 or 1 only defaults are used

2 (ICHINT) indicates what type of integral has to be computed. Possible values:

1 $\int_{\partial\Omega} f u ds$

2 $\int_{\partial\Omega} \mathbf{u}\cdot\mathbf{n} ds$ with $\mathbf{n}$ the normal.

3 $\int_{\partial\Omega} \mathbf{u}\cdot\mathbf{t} ds$ with $\mathbf{t}$ the tangential vector.

4 $\int_{\partial\Omega} \mathbf{n}\cdot\sigma\cdot\mathbf{n} ds$ with $\mathbf{n}$ the normal and σ the stress tensor.

5 $\int_{\partial\Omega} \mathbf{n}\cdot\sigma\cdot\mathbf{t} ds$ with $\mathbf{n}$ the normal, $\mathbf{t}$ the tangential vector and σ the stress tensor.

6 $\int_{\partial\Omega} \mathbf{n}\cdot\sigma ds$, with $\mathbf{n}$ the outward normal, and σ the stress tensor. The result of this integration is a vector.

7 $\int_{\partial\Omega} u\mathbf{n} ds$ with $\mathbf{n}$ the outward normal and u a scalar function. The result of this integration is a vector.

8 $\sum_i f_i u_i$, i.e. the sum over the points at the curves of the function. The result is a scalar. This option is meant in combination with reaction forces, see the Users Manual Section 3.2.14.

The possibilities 2 to 8 have only been implemented for NDIM = 2, i.e. two-dimensional regions.
Default value: 1

3 (ICHFUN) indication of the value of $f(\mathbf{x})$. Possible values:

0 $f = 1$

>0 f is a function of the space:

```
f = func ( ichfun, x, y, z )       (real case)
       or
f = cfunc ( ichfun, x, y, z )      (complex case)
```

See Section 5.5.4 in the Introduction manual.
At this moment only the combination $f \neq 1$ and ichint=1 is permitted.

Default value: 0

4 (IRULE) The numerical integration rule to be carried out depends on IRULE. Possibilities: Possible values: For curves only (i.e. 2D):

1 trapezoid rule, cartesian co-ordinates

2 trapezoid rule, axisymmetric co-ordinates

3 simpson's rule, cartesian co-ordinates

4 simpson's rule, axisymmetric co-ordinates

For surfaces only (i.e. 3D):

1 Newton cotes rule, i.e. integration rule based on the type of approximation used in the element

Default value: 1

5 (IDIM) This parameter indicates if curves (1) or surfaces (2) must be computed.
Default value: NDIM-1.

6 (NUMBER) This parameter indicates which unknown (physical degree of freedom) must be used from the (solution) vector u (IVECTR). If ICHINT = 2, 3 or 7, it is supposed that the degrees of freedom are equal to NUMBER, NUMBER+1 and NUMBER+2 (3D). If ICHINT = 4, 5 or 6, it is supposed that the degrees of freedom are equal to NUMBER, NUMBER+1 and NUMBER+2 (2D) or NUMBER to NUMBER+5 (3D).
Default value: 1

7 (NCURVES) Indicates the number of curves or surfaces in array ICURVS that must be used for the integration.
Default value: 1

RINTBN Not yet used.

KMESH, KPROB, IVECTR See BOUNIN (13.3.1).

ICURVS Integer array containing the curve numbers for the computation of the boundary integral. A negative curve number indicates that the curve must be computed in the reversed direction. The curves including their sign must form one contiguous curve The number of curves is stored in array iintbn.
If the integration must be carried out over surfaces, icurvs must contain the surface sequence numbers.
icurvs is only used if iread = -1.

VALUE Array containing the computed integrals at output. When ICHINT = 1 to 5, VALUE has length 1, otherwise value has length 2.
If the vector to be integrated corresponds to a complex vector, VALUE is a complex array

IREAD Defines if parameters should be read or are given in array IINTBN through the parameter list.
Possible values:

0 The input is read from the standard input file.
For a description of the input the reader is referred to the Users Manual Section 3.2.14.

>0 The input as read from the standard input file by subroutine SEPSTN (4.2.3) is used. The value of iread defines the sequence number.

-1 No input is read, all information from array IINTBN is used.

Input

The arrays KMESH, KPROB and IVECTR must have been filled.
Depending on IREAD IINTBN and ICURVS must have been filled.
IREAD must have a value.

Output

Array VALUE has been filled.

14 Computation of derived quantities

14.1 Introduction

In this chapter we shall describe some subroutines that may be used to compute derived quantities. In fact it concerns all quantities as described in the Users Manual Section 3.2.11.

In this chapter we deal with the following paragraphs:

14.2 treats the present subroutines that may be used to compute derived quantities. At this moment it concerns the subroutines DERIV and DERIVS.

14.3 deals with subroutine DERIVA an old fashioned SEPRAN subroutine to compute derived quantities. This subroutine although not recommended may still be used in your main program.

14.4 describes subroutines that may be used to compute a stream function.

14.2 General subroutine for the computation of derived quantities

In this section we shall describe the following subroutines to compute derived quantities:

DERIV (14.2.1) Computes derived quantities of an input vector, either in the nodal points or in the elements.
Input can only be given through the parameter list.

DERIVS (14.2.2) Has the same task as DERIV, however, DERIVS has some extra possibilities not available to DERIV, like computation of integrals. Furthermore input is given by an input array iinder or read from the standard input file.

14.2.1 Subroutine DERIV

Description

Computes derived quantities of an input vector. These quantities are computed in nodal points, or in elements.
Quantities in common nodes of different elements are averaged.
The derived quantities related to the solution of standard problems are described in the manual "Standard Problems".

Call

```
CALL DERIV ( IINDER, IOUTVC, KMESH, KPROB, INPVEC, IUSER, USER )
```

Parameters

INTEGER IINDER(∗), IOUTVC(5), KMESH(∗), KPROB(∗), INPVEC(5,∗), IUSER(∗)

DOUBLE PRECISION USER(∗)

IINDER Integer array in which the user indicates what computations must be performed by subroutine DERIV.
The entries of IINDER must be filled (by the user) as follows: (If the user sets a value equal to 0, the default value is used)

1 Number of entries filled by the user. If 0 or 1 is given only defaults are used.

2 ICHOICE, indicates the type of output vector.
Possibilities:

- 0 Output vector is of the type of solution vector
- 1 Output vector is a vector of special structure (type 115)
- 2 Output vector is a vector of special structure (type 116), i.e. defined per element. No averaging takes place.
- 3 The output vector is a vector of special structure with sequence number ivec. The output vector is made by adding quantities in common nodal points, hence DERIV acts as if a right-hand-side vector is built.
- 4 The input vector is supposed to be a vector corresponding to quadratic elements, of which only the vertices are filled. With this value of ICHOICE the output vector consists of a vector filled in all points, where the midpoints are created by linear interpolation. In this case it is necessary that there is at least a solution vector or a vector of special structure defined such that one unknown is available in all nodes.

Default value: 0

3 ICOMPL, indicates if the output vector is real (0) or complex (1).
Default value: 0

4 ICHELD, choice parameter for the element subroutines.
Default value: 1

5 IX, choice parameter for the element subroutines.
Default value: 1

6 JDEGFD, choice parameter for the element subroutines.
Default value: 1

7 IVEC, Sequence number for the type of vector of special structure, when ICHOICE = 1, or number of degrees of freedom to be computed per element when ICHOICE = 2.
Default value: 1

8 NUMVEC, Number of vectors that are stored in array INPVEC Compare with subroutine BUILD (7.2) If NUMVEC=0, it is supposed that one vector should be used.
Default value: 0

9 CLEARVECTOR, This parameter indicates whether the output vector must be cleared before adding the new results (0), or that the results are added to the previous vector.
Default value: 0

10 ALLELEMENTS Indication if all standard element sequence numbers are used (0) or not (1) In that case the positions 16 to 15+NELGRP (NELGRP is number of element groups) are considered as an array IELHLP to be filled by the user in the following way: In IELHLP(IELGRP) (1¡=IELGRP¡=NELGRP) the user must store, whether elements with standard element sequence number IELGRP are added to the large matrix and large vector or not. When IELHLP(IELGRP) = 0 the elements with standard element sequence number IELGRP are skipped, when IELHLP(IELGRP) = 1 these elements are added.
Default value: 0

IOUTVC In this array information concerning the computed quantities is stored.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

INPVEC Integer array containing information concerning the input vector(s). For a proper choice in the case of standard problems, consult the manual "Standard Problems".

USER,IUSER In these arrays information concerning coefficients etc. for the differential equation must be stored. For standard problems that are contained in the library, the arrays IUSER and USER must be filled according to fixed rules (see the manual Standard Problems). Subroutines FILCOF (6.2.1) may be used to fill IUSER and USER. When the user uses his own elements he may fill and use IUSER and USER in his own way, since these arrays are only used by the element subroutines.

Input

The arrays KMESH, INPVEC, IINDER and KPROB must have been filled.
If necessary the arrays IUSER and USER must have been filled.

Output

The output vector corresponding to IOUTVC, as well as IOUTVC have been filled.

Subroutines called

Subroutine DERIV calls element subroutines. These may be standard subroutines from the library (type numbers ≥ 100), but also user written element subroutines (type numbers < 100).

Remarks

- For groups of elements with problem type number ITYPE $\leq$ 99, the user must program an element subroutine with the name ELDERV when ICHOICE = 1 or 2, or a routine with the name ELCERV when ICHOICE = 3. See subroutines ELDERV (Users manual 4.5) and ELCERV (Users manual 4.6).
- If during the mesh generation, the COMMAND MESHCONNECT is used, the solution is periodical. If a derived quantity (IOUTVC) is of the same type as the solution vector, then this quantity is assumed to be periodical too, and it is treated as such by the averaging procedure. If, however the derived quantity is not periodical, it is necessary to define a vector of special structure for this vector.
  Even if the derived quantity (IOUTVC) is a vector of special structure, it is treated as a periodical vector, unless a special precaution is taken. In that case it is necessary to exclude the element group corresponding to the periodical boundary condition from the averaging procedure. So ALLELEMENTS = 1 together with array IELHLP should be used.

14.2.2 Subroutine DERIVS

Description

Computes derived quantities of an input vector. These quantities are computed in nodal points, or in elements.
Quantities in common nodes of different elements are averaged.
The derived quantities related to the solution of standard problems are described in the manual "Standard Problems".
Input may be provided through the parameter list but also through the input file.

Call

```
CALL DERIVS ( INPDER, IOUTVC, KMESH, KPROB, INPVEC, IUSER,
              USER, IREAD, OUTPUT, IPROB, ICURVS )
```

Parameters

INTEGER INPDER(∗), IOUTVC(5), KMESH(∗), KPROB(∗), INPVEC(5,∗), IUSER(∗), IREAD(∗), IPROB, ICURVS(∗)

DOUBLE PRECISION USER(∗), OUTPUT(∗)

INPDER Integer array in which the user indicates what computations must be performed by subroutine DERIVS.
inpder is only used if iread(1) = -1 or iread(1)>0 If iread(1)>0,inpder(1) defines the scalar sequence number. This number must be >0 if an integral must be computed. inpder(2) defines numvec.
If iread(1) = -1 then the entries of INPDER must be filled (by the user) as follows (If the user sets a value equal to 0, the default value is used):

1 Number of entries filled by the user

2 ICHOICE, indicates the type of output vector.
Possibilities:

-1 No output vector required.
In this case ITYPE_BOUN must be > 0

0 Output vector is of the type of solution vector.

1 Output vector is a vector of special structure (type 115).

2 Output vector is a vector of special structure (type 116), i.e. defined per element. No averaging takes place.

3 The output vector is a vector of special structure with sequence number ivec. The output vector is made by adding quantities in common nodal points, hence DERIVS acts as if a right-hand-side vector is built.

4 The input vector is supposed to be a vector corresponding to quadratic elements, of which only the vertices are filled. With this value of ICHOICE the output vector consists of a vector filled in all points, where the midpoints are created by linear interpolation. In this case it is necessary that there is at least a solution vector or a vector of special structure defined such that one unknown is available in all nodes.

3 ICOMPL, indicates if the output vector is real (0) or complex (1).

4 ICHELD, choice parameter for the element subroutines.

5 IX, choice parameter for the element subroutines.

6 JDEGFD, choice parameter for the element subroutines.

7 IVEC, Sequence number for the type of vector of special structure, when ICHOICE = 1, or number of degrees of freedom to be computed per element when ICHOICE = 2.

8 NUMVEC, Number of vectors that are stored in array INPVEC Compare with subroutine BUILD (7.2).
If NUMVEC=0, it is supposed that one vector should be used.

9 CLEARVECTOR, This parameter indicates whether the output vector must be cleared before adding the new results (0), or that the results are added to the previous vector.

10 SEQ_INPUTVEC1 Indicates the sequence number of the first vector in array INPVEC.

11 SEQ_INPUTVEC2 Indicates the sequence number of the second vector in array INPVEC.

12 ALLELEMENTS Indication if all standard element sequence numbers are used (0) or not (>0) In that case the positions ALLELEMENTS to ALLELEMENTS+NELGRP-1. (NELGRP is number of element groups) are considered as an array ISKPEL to be filled by the user in the following way: In ISKPEL(IELGRP) (1¡=IELGRP¡=NELGRP) the user must store, whether elements with standard element sequence number IELGRP are added to the large matrix and large vector or not. When ISKPEL(IELGRP) = 0 the elements with standard element sequence number IELGRP are skipped, when ISKPEL(IELGRP) = 1 these elements are added.
Default value: 0

13 ITYPE_BOUN Indicates the type of boundary integral to be computed. Possible values:

0 No boundary integral is computed

2 $\int_\Gamma \mathbf{u} \cdot \mathbf{n} ds$, where $\mathbf{u}$ is the vector just computed.
Hence the derivative must be a vector.
In this case ICHOICE must be -1.

Other values are not yet available.
Default value: 0

14 IRULE_BOUN Indicates the type of integration rule to be applied. Possible values:

1 trapezoid rule, cartesian co-ordinates

Other values are not yet available.
Default value: 0

15 INUMCURVES Is only used if iinder(13)>0.
It indicates the number of curves to be used, See also array ICURVS.

IOUTVC In this array information concerning the computed quantities is stored.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

INPVEC Integer array containing information concerning the input vector(s). For a proper choice in the case of standard problems, consult the manual "Standard Problems".

USER,IUSER In these arrays information concerning coefficients etc. for the differential equation must be stored. For standard problems that are contained in the library, the arrays IUSER and USER must be filled according to fixed rules (see the manual Standard Problems). Subroutines FILCOF (6.2.1) may be used to fill IUSER and USER. When the user uses his own elements he may fill and use IUSER and USER in his own way, since these arrays are only used by the element subroutines.

IREAD Array iread defines if parameters should be read or are given through the parameter list.
iread(1) refers to inpder.
iread(2) refers to iuser/user

iread(1) indicates if array INPDER is used or that input is read from the standard INPUT file.
Possible values:

- 0 The input is read from the standard input file
- >0 The input as read from the standard input file by subroutine SEPSTN (4.2.3) is used. The value of iread defines the sequence number.
- -1 No input is read, all information from array INPDER is used.

iread(2) indicates if the arrays IUSER/USER are used or that input is read from the standard INPUT file
Possible values:

- 0 The input is read from the standard input file
- >0 The input as read from the standard input file by subroutine SEPSTN (4.2.3) is used. The value of iread defines the sequence number of the coefficients.
- -1 No input is read, all information from arrays IUSER/USER is used

OUTPUT Array containing the integrals that are computed if ICHOIS>10

IPROB Actual problem number

ICURVS Integer array containing the curve numbers for the computation of the boundary integral. A negative curve number indicates that the curve must be computed in the reversed direction. The curves including their sign must form one contiguous curve The number of curves is stored in array iinder icurvs is only used if iread(1) = -1

Input

The arrays KMESH, INPVEC, INPDER, IREAD and KPROB must have been filled.
If necessary the arrays ICURVS, IUSER and USER must have been filled.
IPROB must have a value.

Output

The output vector corresponding to IOUTVC, as well as IOUTVC have been filled.
Depending on the input array OUTPUT has been filled.

Subroutines called

Subroutine DERIVS calls element subroutines. These may be standard subroutines from the library (type numbers ≥ 100), but also user written element subroutines (type numbers < 100).

Remarks

The same remarks as for DERIV are valid.

14.3 Old fashioned subroutines to compute derived quantities(DERIVA)

Subroutine DERIVA *Description*

Computes derived quantities of an input vector. Depending on the value of ICHOICE, quantities are computed in nodal points, or in elements.
Quantities in common nodes of different elements are averaged.
The derived quantities related to the solution of standard problems are described in the manual "Standard Problems".

Call

```
CALL DERIVA ( ICHOICE, ICHELD, IX, JDEGFD, IVEC, IOUTVC, KMESH, KPROB,
              INPVC0, INPVC1, IUSER, USER, IELHLP )
```

Parameters

INTEGER ICHOICE, ICHELD, IX, JDEGFD, IVEC, IOUTVC(5), KMESH(∗), KPROB(∗), INPVC0(5), INPVC1(5), IUSER(∗), IELHLP(∗)

DOUBLE PRECISION USER(∗)

ICHOICE Choice parameter indicating which is the type of the output vector. Possibilities:

1 the output vector is of the same type as the solution vector. Quantities in common nodal points are averaged.

2 the output vector is a vector of special structure with sequence number IVEC (see input block PROBLEM). Quantities in common nodal points are averaged.

3 the output vector is defined per element. No averaging takes place.

4 The output vector is of special structure with sequence number IVEC (see input block PROBLEM). The output vector is made by adding quantities in common nodal points, hence in this case DERIVA reacts as if it builds a right-hand-side vector. However, this right-hand-side vector is not of the type of the solution vector, but a vector of special structure, and in that sense DERIVA differs from BUILD (7.2).

11-14 See ICHOICE = 1-4 respectively. However, the user must indicate which element groups are taken into account and which element groups are skipped. Array IELHLP is used for this determination.

ICHELD Choice parameter to indicate which of the possible derived quantities has to be computed. ICHELD is used in the element subroutines. Consult the manual "Standard Problems" for a proper selection of ICHELD, in the case of type numbers larger than 99.
For user defined problems (type numbers < 100), the selection of ICHELD is free. See the description of subroutine ELDERV (Users Manual 4.5)

IX Sequence number of space variable.
i.e. IX = 1 corresponds to x, IX = 2 to y, etc.

JDEGFD Sequence number of degrees of freedom in a nodal point. The $JDEGFD^{th}$ degree of freedom is used to compute the derivative.

IVEC When IVEC < 0 the output vector is a complex vector, otherwise when IVEC ≥ 0, it is a real one.
When ICHOICE = 1, only the sign of IVEC is used.
When ICHOICE = 2, |IVEC| gives the sequence number of the output array of special structure.
When ICHOICE = 3, |IVEC| gives the number of degrees of freedom to be computed in all element groups.

IOUTVC In this array information concerning the computed quantities is stored.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

INPVC0,INPVC1 Integer array containing information concerning the input vector(s). For a proper choice in the case of standard problems, consult the manual "Standard Problems". Generally only INPVC0 is used, however, for some applications INPVC1 is also necessary. When both arrays are used they must be of the same type.
Remark: When INPVC1 is not used, INPVC0 must be used as actual parameter in INPVC1.

USER,IUSER In these arrays information concerning coefficients etc. for the differential equation must be stored. For standard problems that are contained in the library, the arrays IUSER and USER must be filled according to fixed rules (see the manual Standard Problems). Subroutines FILCOF (6.2.1) may be used to fill IUSER and USER. When the user uses his own elements he may fill and use IUSER and USER in his own way, since these arrays are only used by the element subroutines.

IELHLP Help array to be used when ICHOICE = 11,12,13 or 14.
In IELHLP(IELGRP + 5) ($1 \leq$ IELGRP $\leq$ NELGRP+NUMNATBND) the user must store, whether elements with standard element sequence number IELGRP (or IELGRP = NELGRP + IBNGRP for boundary elements) are added to the large matrix and large vector or not.
When IELHLP(IELGRP+5) = 0 the elements with standard element sequence number IELGRP are skipped, when IELHLP(IELGRP+5) = 1 these elements are added.
IELHLP is a variable length array, hence position 1 must be filled by the user.

Input

The parameters ICHOICE, IVEC, and if necessary ICHELD, IX, and JDEGFD must have a value.
The arrays KMESH, and KPROB must have been filled.
If necessary the arrays IELHLP, IUSER and USER must have been filled.
Array INPVC0 and if necessary array INPVC1 must have been filled.

Output

The output vector corresponding to IOUTVC, as well as IOUTVC have been filled.

Subroutines called

Subroutine DERIVA calls element subroutines. These may be standard subroutines from the library (type numbers ≥ 100), but also user written element subroutines (type numbers < 100).

Remarks

- For groups of elements with problem type number ITYPE ≤ 99, the user must program an element subroutine with the name ELDERV when ICHOICE = 1 or 2, or a routine with the name ELCERV when ICHOICE = 3. See subroutines ELDERV (Users manual 4.5) and ELCERV (Users manual 4.6).
- If during the mesh generation, the COMMAND MESHCONNECT is used, the solution is periodical. If a derived quantity (IOUTVC) is of the same type as the solution vector, then this quantity is assumed to be periodical too, and it is treated as such by the averaging procedure. If, however the derived quantity is not periodical, it is necessary to define a vector

of special structure for this vector.
Even if the derived quantity (IOUTVC) is a vector of special structure, it is treated as a periodical vector, unless a special precaution is taken. In that case it is necessary to exclude the element group corresponding to the periodical boundary condition from the averaging procedure. So ICHOICE = 12, together with array IELHLP should be used.

14.4 Computation of the stream function

At this moment the stream function may be computed by the post-processing program SEPPOST or directly by a call to a subroutine.
There are two subroutines for the computation of stream functions available:

STREAM (14.4.1) Standard subroutine for the computation of a stream function.

STRM1 (14.4.2) Extension of STREAM for the case that one or more element groups must be skipped.

14.4.1 Subroutine STREAM

Description

Compute the stream function ψ or potential ϕ of a two-dimensional vector **v** for various co-ordinate systems.

Heading

```
subroutine stream ( ichels, ivec, istrm, jpoint, psiphi, kmesh, kprob, ivectr )
```

Parameters

INTEGER ICHELS, IVEC, ISTRM(5), JPOINT, KMESH(∗), KPROB(∗), IVECTR(∗)

DOUBLE PRECISION PSIPHI

ICHELS Choice parameter, defining the type of function to be evaluated.
Consult the manual "Standard Problems" for the correct value of ICHELS.
Generally ICHELS = 1 corresponds to ψ and ICHELS = 2 to ϕ. If in a mesh different types of elements are used, ICHELS and IVEC must be equal for all these elements.
For user defined elements (type numbers < 100), the selection of ICHELS is free. See the description of subroutine ELSTRM.

IVEC This parameter indicates the type of output vector. The output vector is always a vector of special structure with sequence number IVEC.
When elements of type number larger than 100 are used, then IVEC is computed automatically even when these elements are combined with user defined elements. When all elements are user defined (type numbers < 100), the user must specify IVEC himself.

ISTRM Integer array of length 5 containing information concerning the stream-function values or the potential.

JPOINT The output vector (stream-function or potential) is fixed except for an additive constant. The user may fix this constant in the **vertex** of an element by giving JPOINT the value of the nodal point number of this vertex. This point must lie within an element of an accessible element group. When JPOINT = 0, the output vector gets value 0 in nodal point 1.

PSIPHI This parameter is only used when JPOINT > 0. In that case the vertex JPOINT gets the value PSIPHI.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

IVECTR Integer array containing information of the vector from which the stream-function or potential is computed. In general IVECTR is a vector containing information concerning the velocities. Usually IVECTR is the vector ISOL.

Input

The arrays KMESH, KPROB and IVECTR must have been filled.
The parameters ICHELS, JPOINT and PSIPHI must have a value.

Output

Array ISTRM has been filled.

Subroutines called

Subroutine STREAM calls element subroutines.
These may be standard subroutines from the library (type numbers ≥ 100), but also user written element subroutines (type numbers < 100).

Remarks

- For groups of elements with problem type number ITYPE ≤ 99, the user must program an element subroutine with the name ELSTRM. See subroutine ELSTRM (Users Manual, Section 4.8).
- The stream function is only defined for a vector field $\mathbf{v}$ that is divergence-free, i.e. $div\ \mathbf{v} = 0$. In that case ψ is defined by:

$$\frac{\partial \psi}{\partial x} = -v_2 \qquad \frac{\partial \psi}{\partial y} = v_1 \tag{14.4.1}$$

for cartesian co-ordinates,

$$\frac{\partial \psi}{2\pi r \partial z} = v_r \qquad \frac{\partial \psi}{2\pi r \partial r} = -v_z \tag{14.4.2}$$

for axi-symmetrical co-ordinates and

$$\frac{\partial \psi}{r \partial \phi} = v_r \qquad \frac{\partial \psi}{\partial r} = -v_\phi \tag{14.4.3}$$

for polar co-ordinates.
The potential ϕ is only defined for a vector field $\mathbf{v}$ that is irrotational, i.e. $\mathbf{curl}\ \mathbf{v} = \mathbf{0}$.
In that case ϕ is defined by: $\mathbf{v} = \nabla\phi$.
The factor 2π for axi-symmetrical co-ordinates ensures that ψ measures the volume flux.

14.4.2 Subroutine STRM1

Description

See subroutine STREAM (14.4.1), however, in this case it is allowed to skip some element groups from the computation.

Heading

```
subroutine strm1 ( ichoice_strm, ichels, ivec, istrm, jpoint, psiphi, kmesh, kprob,
                   ivectr, ielhlp )
```

Parameters

INTEGER ICHOICE_STRM, ICHELS, IVEC, ISTRM(5), JPOINT, KMESH(∗), KPROB(∗), IVECTR(∗), IELHLP(∗)

DOUBLE PRECISION PSIPHI

ICHOICE_STRM Indication whether element groups must be skipped (ICHOICE_STRM = 1), or not
(ICHOICE_STRM = 0). Hence ICHOICE_STRM = 0 gives exactly the same result as subroutine STREAM.

ICHELS, IVEC, ISTRM, JPOINT, PSIPHI, KMESH, KPROB, IVECTR See subroutine STREAM (14.4.1).

IELHLP Help array to be used when ICHOIS = 1.
In IELHLP(IELGRP + 5) (1 ≤ IELGRP ≤ NELGRP+NUMNATBND) the user must store, whether elements with standard element sequence number IELGRP are used for the computation of the stream function or not.
When IELHLP(IELGRP+5) = 0 the elements with standard element sequence number IELGRP are skipped, when IELHLP(IELGRP+5) = 1 these elements are used.
The consequence of skipping some element groups is that in some part of the domain the stream function is not computed.
IELHLP is a variable length array, hence position 1 must be filled by the user.

Input

The arrays KMESH, KPROB and IVECTR must have been filled.
The parameters ICHOICE_STRM, ICHELS, JPOINT and PSIPHI must have a value.
If ICHOICE_STRM = 1, array IELHLP must have been filled

Output

Array ISTRM has been filled.

Subroutines called

See subroutine STREAM (14.4.1).

15 Print output

15.1 Introduction

In this section we shall describe some subroutines that may be used to print results in a user written program. Some of the subroutines described in this chapter are part of the SEPRAN post processor SEPPOST and the SEPRAN computational program SEPCOMP.

The print subroutines write their output to the file indicated in the SEPRAN environment file sepran.env. See the Users Manual Section 1.5 for details. If the output file is standard output without file name, this means that the output is written to the screen. The output may be saved by redirecting the output to a file, for example by

```
sepcomp < inputfile > outputfile
```

Internally the output is written to the file with reference number irefwr, as stored in common block CMCDPI. This common block is described in Section 21.5. The parameter irefwr is made equal to the parameter *unit number for writing of SEPRAN input* in the sepran.env file (Users Manual Section 1.5).
If the user wants to write the output of a print subroutine to another file it is sufficient to open this file with some reference number and then give irefwr the value of this reference number.
After the call of the print subroutine it is advised to reset the value of irefwr to its original value since otherwise from that moment on, all output including error messages is written to the new file.

In this chapter we deal with the following paragraphs:

15.2 treats how to print solution vectors or vectors containing derived quantities. The whole vector may be printed, but also a restricted part, for example a closed region or a part of the boundary. This printing is performed by the subroutine PRINVC.

15.3 deals with the printing of the structure of the problem. All information about the mesh (coordinates, topology and so on), the problem description or the structure of the matrices may be printed. Subroutine PRINI2 performs this task.

15.4 describes the matrix print subroutine PRINMT.

15.5 describes some subroutines to print selected items of the mesh.

15.2 Printing of solution vectors (subroutine PRINVC)

In this section we shall describe the general SEPRAN subroutine PRINVC to print solution vectors and vectors of special structure. This subroutine may also be used to print the right-hand side vector since this vector has the same structure as the solution vector.
In previous versions of SEPRAN several print subroutines were available:

PRINRV (15.2.2) Print a SEPRAN array with reals or complex quantities. This subroutine provides very little extra information.

PRINOV (15.2.3) Print a SEPRAN array with reals or complex quantities. Nodes are ordered with increasing sequence of the co-ordinates. The printing may be restricted to a rectangular region.

PRINTV (15.2.4) Print values of a solution array or array of special structure in the same way as in PRINOV. No selecting mechanism is available hence the number of parameters is minimal.

PRINCR (15.2.5) Print values of a solution array or array of special structure along a series of user curves.

PRINBN (15.2.6) Print values of a solution array or array of special structure along a series of user curves or surfaces. Extension of PRINCR.

These subroutines are treated at the end of this section. Subroutine PRINVC contains all the possibilities these old subroutines have.

15.2.1 Subroutine PRINVC

Description

General print subroutine. The values of a SEPRAN solution vector or vector of special structure are printed.
The printing may be reduced to printing along curves or surfaces. If necessary the tangential or normal component along these curves or surfaces is printed.
The user may restrict the print region to a rectangular region.
The nodes are printed in increasing sequence of the co-ordinates. Which co-ordinates runs the fastest may be influenced by the user.

Heading

```
subroutine prinvc ( inppri, kmesh, kprob, ivectr, icurvs, text )
```

Parameters

INTEGER IINPRI(∗), KMESH(∗), KPROB(∗), IVECTR(∗), ICURVS(∗)

CHARACTER ∗(∗) TEXT

IINPRI General integer input array. With this array the user may make some choices of how he wants to print the contents of the arrays. The length of the array depends on the number of entries that have been filled by the user. At present only 8 entries have been defined, but in the future this number may be extended. However, the construction is such that there is no need to adapt programs using PRINVC in case of future releases unless the user wants to utilize the extra options then provided. The entries of IINPRI must be filled by the user as follows:

1: LAST Number of entries of IINPRI that are filled by the user. IINPRI(1) may be 0 or 1 in which case only default values are used. For all positions beyond position IINPRI(1), defaults are used. If a user fills an entry of IINPRI with a zero, also the default value is used.

2: ICHOICE Indicates the type of print to be made.
ICHOICE consists of two parts ICHOICE_orig and ISUPPRESS according to ICHOICE = ICHOICE_orig + 10 × ISUPPRESS.
These parameters have the following options:
ICHOICE_orig

0: The nodal point number is printed followed by the values in the nodal point.

1: The nodal point number is printed followed by the co-ordinates and then the values in the nodal point.

2: The nodal point number is printed followed by the co-ordinates.

3: See 2, however, preceded by the number of nodal points to be printed

Default value: 0

ISUPPRESS

0: The nodal point number is printed.

1: The nodal point number is suppressed.

Default value: 0

3: IDIM Indicates which quantities must be printed. Possible values:

0: The complete vector is printed

1: The vector is only printed in user points. These user points must be defined in array ICURVS.

2: The vector is only printed in user curves. These curves must be defined in array ICURVS.

3: The vector is only printed in user surfaces. These surfaces must be defined in array ICURVS.

4: The vector is only printed in user volumes. These volumes must be defined in array ICURVS.

Default value: 0

4: IQUANTITY Indicates which quantity must be printed. Possible values:

0: The contents of the vector are printed as stored in the vector corresponding to IVECTR i.e. without transformation.

1: The normal component of a vector along the curve or surface is printed. (only if IDIM>1).

2: The tangential component(s) of a vector along the curve or surface is printed. (only if IDIM>1).

Default value: 0

5: NUMBER Indicates which degree of freedom in each point must be printed. This parameter must be seen in connection with IQUANTITY. Possible values:

0: If iquantity=0, all degrees of freedom are printed in the nodes defined by icurvs, if iquantity>0, the vector from which the normal or tangential components must be computed is supposed to be a n-dimensional vector with first degree of freedom 1, second degree of freedom 2 and third degree of freedom 3. In this case it is necessary that IDIM = NDIM (dimension of space)

i>0: If iquantity=0, the degree of freedom i is printed. If iquantity>0, the vector from which the normal or tangential components must be computed

is supposed to be a n-dimensional vector with first degree of freedom i, second degree of freedom i+1 and third degree of freedom i+2. In this case it is necessary that IDIM = NDIM (dimension of space)

Default value: 0

6: ISEQ Indicates the sequence in which the co-ordinates along the surface must be ordered in case idim=0 or idim=3. Possible values:

1: increasing x, for fixed x increasing y, for fixed y increasing z.

2: increasing y, for fixed y increasing z, for fixed z increasing x.

3: increasing z, for fixed z increasing x, for fixed x increasing y.

4: sequence of nodal point numbers.

Default value: 1

7: IREGION Indicates if the complete region must be printed (0) or that the printing is restricted to a subregion, defined by the parameters xmin, xmax, ymin, ymax, zmin, zmax in the common block CPLOT. For a description of CPLOT see Section 21.11. Possible values:

0: The complete region is printed

1: The region is restricted to xmin≤x≤xmax, ymin≤y≤ymax, zmin≤z≤zmax.

-1: The region is restricted to x=xmin.

-2: The region is restricted to y=ymin.

-3: The region is restricted to z=zmin.

-11: The region is restricted to x≥xmin.

-12: The region is restricted to y≥ymin.

-13: The region is restricted to z≥zmin.

-21: The region is restricted to x≤xmin.

-22: The region is restricted to y≤ymin.

-23: The region is restricted to z≤zmin.

Default value: 0

If the user uses IREGION ≠ 0 he must submit the common block CPLOT in the program or subroutine that calls PRINVC. This may be done by exactly typing the text as given in Section 21.11, or by copying the common block from the directory SPHOME/common, where SPHOME stands for the SEPRAN home directory.

8: IHEADER Indicates if the header must be printed (0) or not (1)

Default value: 0

9: ICOOR type of co-ordinate system Possible values:

1 Cartesian (x,y)

3 Polar (r, phi)

Default value: 1

10: IFORMAT Format used for printing Possible values:

1 Reals are printed in format e13.5

2 Reals are printed in format e16.8

3 Reals are printed in format e13.5, one node per line

4 Reals are printed in format e16.8, one node per line

Default value: 1

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

KPROB Standard SEPRAN array containing information about the problem description. KPROB must have been filled before for example by a SEPRAN start routine that reads the mesh.

IVECTR Integer SEPRAN array of length 5 corresponding to a solution vector or a vector of special structure. Since the right-hand side vector has the same structure as the solution vector, IVECTR may also be the vector IRHSD.
The array related to IVECTR is printed by this subroutine.

ICURVS Integer array that must be defined and declared by the user.
ICURVS must contain the user points, curve or surface numbers that must be printed in case IDIM>0. If IDIM=0 array ICURVS is not used.
Position 1 indicates how array icurvs is filled. Possible values:

- 0 only one curve is used. The curve number is stored in icurvs(2).
- -1 The solution is printed along the curves icrv1 to icrv2. icrv1 is stored in icurvs(2) and icrv2 in icurvs(3).
- -2 See -1, however, the sequence of the printing is reversed. In case of curves this means that the curves are printed from the end point of icrv2 to the initial point of icrv2, then the end point of icrv2-1 to the begin point of icrv2-1 and so on.
  In case of user points or surfaces ichois=-2 is treated as ichois=-1
- n>0 The curve numbers numbers are stored in icurvs(2) to icurvs(n+1). That sequence is used.

Remarks: a negative curve number means that the curve is printed from end point to initial point, hence in reversed order. Negative user point or surface numbers are not yet available.

Everywhere where curves are mentioned, either user points, curves or surfaces are meant.

TEXT Text to be printed as header for the output. This causes the following text to be printed, as header for the output:

VECTOR PRINTOUT "TEXT", with "TEXT" the actual value of the parameter.
Examples are: 'SOLUTION', 'COORDINATES', 'U-VELOCITY'.

Input

Array IINPRI must have been filled by the user.
The arrays KMESH, KPROB and IVECTR must have been filled.
Depending on the parameter IDIM array ICURVS must have been filled.
Depending on the parameter HEADING, TEXT must have a value.
Depending on the parameter IREGION, common block CPLOT must have been filled by the user.

Output

Print of the vector.

Remarks

An alternative of subroutine PRINVC is subroutine PRINVCBF. This subroutine is identical to PRINVC, except for two extra parameters IBUFFR and BUFFER, that are used as first and second parameter. In this way the local use of blank common is avoided.

Heading:

```
subroutine prinvcbf ( ibuffr, buffer, inppri, kmesh, kprob,
                      ivectr, icurvs, text )
```

Subroutines used in previous versions of SEPRAN

As indicated in the introduction of this section in previous versions of SEPRAN the subroutines PRINRV, PRINOV, PRINTV, PRINCR and PRINBN have been used for the printing of vectors. Here we shall give a short description of these subroutines. Although not longer recommended they still may be used in future versions of SEPRAN.

15.2.2 Subroutine PRINRV

Description

A SEPRAN array with reals (coordinates, solution vector, right-hand side vector or array of related quantities) is printed. At the users choice the whole array is printed or only a special unknown.

Heading

```
subroutine prinrv ( ivectr, kmesh, kprob, layout, number, text )
```

Parameters

INTEGER IVECTR(∗), KMESH(∗), KPROB(∗), LAYOUT, NUMBER

CHARACTER ∗(∗) TEXT

IVECTR Integer SEPRAN array of length 5 corresponding to a solution vector or a vector of special structure.
The array related to IVECTR is printed by this subroutine.

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

KPROB Standard SEPRAN array containing information about the problem description. KPROB must have been filled before for example by a SEPRAN start routine that reads the mesh.

LAYOUT Parameter that indicates what kind of layout is used. Possibilities:

1 The array is printed element after element, 5 in each line. Format: 5E14.6

2 The array is printed element after element, 10 in each line. Format: 10E11.3

3 The array is printed element after element, 10 in each line. Format: 10F11.3

4 Each nodal point (or element when IVECTR is a vector of special structure defined per element) starts on a new line. Format: 5E14.6

5 When only one unknown is required in each nodal point (or element, see 4) (NUMBER > 0) this possibility gives 5 elements in each line each provided with the corresponding nodal point number. Format: 5E11.3

6 See 5, however Format: 5F11.3

7,8 See 5,6, however, only 4 elements in each row. This possibility is made for 80 columns output (terminals and small printers).

The values LAYOUT = 1, 4, 7 and 8 are meant for 80 columns output, the other ones require 120 columns output.

NUMBER Choice parameter. Possibilities:

-1 The coordinates are printed, array IVECTR is not used.

0 The array corresponding to IVECTR is printed entirely.

>0 Only the $NUMBER^{th}$ unknown in each nodal point of the vector of which the information is stored in array IVECTR, is printed. When the unknown does not appear in a nodal point this point is skipped.
When array IVECTR corresponds to a complex vector then NUMBER = 1 corresponds to the real part of the first unknown in each nodal point and NUMBER = 2 to the imaginary part. In the same way NUMBER = 3 corresponds to the real part of the second unknown etc. Hence when the real part of the i^{th} unknown is required, NUMBER should have the value 2 × i - 1, for the imaginary part NUMBER should be equal to 2 × i.

TEXT Text to be printed as header for the output.

Input

The parameters LAYOUT, NUMBER and TEXT must have a value.
The arrays KMESH, KPROB and IVECTR must have been filled.

Output

Print of the vector.

Remarks

An alternative of subroutine PRINRV is subroutine PRINRVBF. This subroutine is identical to PRINRV, except for two extra parameters IBUFFR and BUFFER, that are used as first and second parameter. In this way the local use of blank common is avoided.

Heading:

```
subroutine prinrvbf ( ibuffr, buffer, ivectr, kmesh, kprob,
                      layout, number, text )
```

15.2.3 Subroutine PRINOV

Description

A SEPRAN array with real or complex values (solution vector, right-hand-side vector or array of related quantities) is printed. At the users choice the whole array is printed or only a special unknown. Each node starts on a new line and the co-ordinates of the nodes are printed as well. The nodes are sorted for increasing x, y or z-co-ordinates. At the users choice a part of the region may be printed.

Heading

```
subroutine prinov ( ivectr, kmesh, kprob, jdim, text, xjdim, jpoint )
```

Parameters

INTEGER IVECTR(∗), KMESH(∗), KPROB(∗), JDIM, JPOINT

CHARACTER ∗(∗) TEXT

DOUBLE PRECISION XJDIM(∗)

IVECTR Integer SEPRAN array of length 5 corresponding to a solution vector or a vector of special structure.
The array related to IVECTR is printed by this subroutine.

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

KPROB Standard SEPRAN array containing information about the problem description. KPROB must have been filled before for example by a SEPRAN start routine that reads the mesh.

JDIM With the aid of JDIM both the ordering of the nodes and the part of the region to be printed is selected. Permitted values of JDIM are:

- 0 The complete region is printed. The nodes are ordered in first instance with respect to increasing x_1-co-ordinate then with respect to increasing x_2-co-ordinate and finally with respect to increasing x_3-co-ordinate.
  The heading above the print is suppressed.
- 1 The complete region is printed. The nodes are ordered in first instance with respect to increasing x_1-co-ordinate then with respect to increasing x_2-co-ordinate and finally with respect to increasing x_3-co-ordinate.
- 2 The complete region is printed. The nodes are ordered in first instance with respect to increasing x_2-co-ordinate then with respect to increasing x_3-co-ordinate and finally with respect to increasing x_1-co-ordinate.
- 3 The complete region is printed. The nodes are ordered in first instance with respect to increasing x_3-co-ordinate then with respect to increasing x_1-co-ordinate and finally with respect to increasing x_2-co-ordinate.
- − See JDIM = n, however only the points with x_{JDIM} = XJDIM(i), (i = 1 (1) jpoint) are selected.
- $10 + n$ See JDIM = n, however only the part of the region given by $x_n \geq$ XJDIM(1) is printed.
- $-10 - n$ See JDIM = n, however only the part of the region given by $x_n \leq$ XJDIM(1) is printed.
- $20 + n$ See JDIM = n, however only the part of the region given by $XJDIM(1) \leq x_n \leq XJDIM(2)$ is printed.

$30 + n$ See JDIM $= n$, however only the part of the region given by $XJDIM(1) \leq x_1 \leq XJDIM(2)$, $XJDIM(3) \leq x_2 \leq XJDIM(4)$, $XJDIM(5) \leq x_3 \leq XJDIM(6)$ is printed.

TEXT Text to be printed as header for the output.

XJDIM This array is used depending on the value of JDIM. See JDIM for an explanation.

JPOINT See JDIM for its use.

Input

The arrays IVECTR, KMESH and KPROB must have been filled.
Depending on the value of JDIM array XJDIM must have been filled.
JDIM and TEXT must have a value.
Depending on the value of JDIM JPOINT must have a value.

Output

Print of the vector.

Remarks

An alternative of subroutine PRINOV is subroutine PRINOVBF. This subroutine is identical to PRINOV, except for two extra parameters IBUFFR and BUFFER, that are used as first and second parameter. In this way the local use of blank common is avoided.

Heading:

```
subroutine prinovbf ( ibuffr, buffer, ivectr, kmesh, kprob, jdim,
                      text, xjdim, jpoint )
```

15.2.4 Subroutine PRINTV

Description

Subroutine PRINTV performs the same task as subroutine PRINOV (15.2.3), however without the possibility to define the region or the sequence of the nodal points.
As a consequence the number of parameters is less than for PRINOV and for a simple print PRINTV is preferred.

Heading

```
subroutine printv ( ivectr, kmesh, kprob, text )
```

Parameters

INTEGER IVECTR(∗), KMESH(∗), KPROB(∗)

CHARACTER ∗(∗) TEXT

IVECTR Integer SEPRAN array of length 5 corresponding to a solution vector or a vector of special structure.
The array related to IVECTR is printed by this subroutine.

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

KPROB Standard SEPRAN array containing information about the problem description. KPROB must have been filled before for example by a SEPRAN start routine that reads the mesh.

TEXT Text to be printed as header for the output.

Input

The arrays IVECTR, KMESH and KPROB must have been filled. TEXT must have a value.

Output

Print of the vector.

Remarks

An alternative of subroutine PRINTV is subroutine PRINTVBF. This subroutine is identical to PRINTV, except for two extra parameters IBUFFR and BUFFER, that are used as first and second parameter. In this way the local use of blank common is avoided.

Heading:

```
subroutine printvbf ( ibuffr, buffer, ivectr, kmesh, kprob, text )
```

15.2.5 Subroutine PRINCR

Description

Subroutine PRINCR may be used to print values of an array along curves indicated by the user.

Heading

```
subroutine princr ( ichois, kmesh, kprob, ivectr, number, icurvs, text, ioutput )
```

Parameters

INTEGER ICHOICE, KMESH(∗), KPROB(∗), IVECTR(∗), NUMBER, ICURVS(∗), IOUTP(∗)

CHARACTER ∗(∗) TEXT

ICHOICE Choice parameter consisting of two parts ICHOICE_ORIG and IFORMAT according to ICHOICE = ICHOICE_ORIG + 10 × (IFORMAT-1).
Possible values for ICHOICE_ORIG

0 The nodal point number is printed followed by the values in the nodal point.

1 The nodal point number is printed followed by the co-ordinates and then the values in the nodal point.

Possible values for IFORMAT

1 Reals are printed in format e13.5

2 Reals are printed in format e16.8

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

KPROB Standard SEPRAN array containing information about the problem description. KPROB must have been filled before for example by a SEPRAN start routine that reads the mesh.

IVECTR Integer SEPRAN array of length 5 corresponding to a solution vector or a vector of special structure.
The array related to IVECTR is printed by this subroutine.

NUMBER The $NUMBER^{th}$ degree of freedom in each nodal point with respect to the array corresponding to $IVECTR$ is printed.
When NUMBER = 0 all degrees of freedom in the nodal points along the curves are printed.

ICURVS See subroutine PRINVC (15.2.1).

TEXT Text to be printed as header for the output.

IOUTP Dummy parameter that is not actually used.

Input

The arrays IVECTR, KMESH, KPROB and ICURVS must have been filled. ICHOICE, NUMBER and TEXT must have a value.

Output

Print of the vector along the curves.

Remarks

An alternative of subroutine PRINCR is subroutine PRINCRBF. This subroutine is identical to PRINCR, except for two extra parameters IBUFFR and BUFFER, that are used as first and second parameter. In this way the local use of blank common is avoided.

Heading:

```
subroutine princrbf ( ibuffr, buffer, ichois, kmesh, kprob,
                      ivectr, number, icurvs, text, ioutput )
```

15.2.6 Subroutine PRINBN

Description

Subroutine PRINBN may be used to print values of an array along curves or surfaces indicated by the user.
If necessary the tangential or normal component is printed
Actually PRINBN has been extended such that it is identical to subroutine PRINVC (15.2.1).

Heading

```
subroutine prinbn ( inppri, kmesh, kprob, ivectr, icurvs, text )
```

Parameters

INTEGER IINPRI(∗), KMESH(∗), KPROB(∗), IVECTR(∗), ICURVS(∗)
CHARACTER ∗(∗) TEXT

For a description of the parameters, input and output see subroutine PRINVC (15.2.1).

15.3 Printing of some arrays defining the structure of the problem (subroutine PRINI2)

Description

Subroutine to print the contents of some of the integer arrays describing the mesh, problem definition or the storage of the matrices.
If this subroutine is used, some knowledge of the arrays described in Section 24 is supposed.

Call

```
CALL PRINI2 ( KMESH_KPROB, IVECTR, ICHOICE )
```

Parameters

INTEGER KMESH_KPROB(∗), IVECTR(∗), ICHOICE

KMESH_KPROB Help array that is necessary to interpret the contents of array IVECTR. Usually KMESH_KPROB is either KMESH or KPROB. Which array is necessary depends on the value of ICHOICE.

IVECTR Integer array that contains information about the array or parts of the array to be printed. Which array must be used as actual parameter for IVECTR depends on ICHOICE.

ICHOICE Choice parameter indicating which part of which array is printed. Possible values:

-1 Print a selected set of all parts of the complete integer array IVECTR. This possibility is only allowed for IVECTR equal to KMESH, KPROB or INTMAT.
If the array KMESH must be printed, then both arrays KMESH_KPROB and IVECTR are assumed to be KMESH.
If the array KPROB must be printed, then array KMESH_KPROB is assumed to be KMESH and IVECTR is assumed to be KPROB.
If the array INTMAT must be printed, then array KMESH_KPROB is assumed to be KPROB and IVECTR is assumed to be INTMAT.

0 Print the complete integer array IVECTR, with all corresponding parts. This possibility is only allowed for IVECTR equal to KMESH, KPROB or INTMAT.
The same rules as for ICHOICE = -1 apply.

1 Print the elements with corresponding nodal point numbers (KMESH parts c and g).
KMESH_KPROB must be the array KMESH.
IVECTR is not used.

2 Print all elements corresponding to a nodal point, for each nodal point. (KMESH part d)
KMESH_KPROB must be the array KMESH.
IVECTR is not used.
This possibility has not been programmed yet.

3 Print the neighbour points of the nodal points. (KMESH part e).
KMESH_KPROB must be the array KMESH.
IVECTR is not used.

4 Print the new numbering of the nodal points (KMESH part j).
KMESH_KPROB must be the array KMESH.
IVECTR is not used.

5 Print information of the points, curves and surfaces. (KMESH part h).
KMESH_KPROB must be the array KMESH.
IVECTR is not used.

6 Print the nodal point numbers of the user points (KMESH part l), the nodal point numbers of the curves (KMESH part m) and the elements of the surfaces (KMESH part n).
KMESH_KPROB must be the array KMESH.
IVECTR is not used.

7 Print the neighbour elements of the elements (KMESH part o),
KMESH_KPROB must be the array KMESH.
IVECTR is not used.

8 Print the levels in the mesh (KMESH part k),
KMESH_KPROB must be the array KMESH.
IVECTR is not used.

9 Print information of the curves and surfaces (KMESH parts q, r and s)
KMESH_KPROB must be the array KMESH.
IVECTR is not used.

11 Print the boundary elements with corresponding nodal point numbers. (KPROB parts b and i)
KMESH_KPROB is not used.
IVECTR must be the array KPROB.

12 Print the number of degrees of freedom stored in the nodal points. (KPROB part f)
KMESH_KPROB must be the array KMESH.
IVECTR must be the array KPROB.

13 Print information about the arrays of special structure. (KPROB part e)
KMESH_KPROB must be the array KMESH.
IVECTR must be the array KPROB.

14 Print the internal numbering of the degrees of freedom, after renumbering due to the repositioning of nodal points, and prescribed degrees of freedom. (KPROB part h)
KMESH_KPROB is not used.
IVECTR must be the array KPROB.

15 Print the nodal point sequence numbers of the boundary elements. (KPROB part i)
KMESH_KPROB is not used.
IVECTR must be the array KPROB.

16 Print the number of nodal points in the super elements. (KPROB part k)
KMESH_KPROB is not used.
IVECTR must be the array KPROB.

17 Print the number of entries in the super element matrices. (KPROB part l)
KMESH_KPROB is not used.
IVECTR must be the array KPROB.

18 Print the element numbers of elements in the super elements. (KPROB part m)
KMESH_KPROB is not used.
IVECTR must be the array KPROB.

19 Print information about the local transformations. (KPROB part n)
KMESH_KPROB is not used.
IVECTR must be the array KPROB.

20 Print information concerning the structure of the large matrix corresponding to the internal degrees of freedom. (INTMAT part 1)
KMESH_KPROB must be the array KPROB.
IVECTR must be the array INTMAT.

21 Print information concerning the structure of the large matrix corresponding to the prescribed degrees of freedom. (INTMAT part 2)
KMESH_KPROB must be the array KPROB.
IVECTR must be the array INTMAT.

22 Print information about boundary conditions of the type u = constant. (KPROB part o)
KMESH_KPROB is not used.
IVECTR must be the array KPROB.

23 Print information about the renumbering of unknowns when physical variables are used or more than one unknown is available (KPROB part p)
KMESH_KPROB is not used.
IVECTR must be the array KPROB.

24 Print information about the internal renumbering of the unknowns with respect to the matrices (KPROB part r).
KMESH_KPROB is not used.
IVECTR must be the array KPROB.

30 Print the arrays IINPUT and RINPUT from array KMESH.
KMESH_KPROB must be equal to array KMESH.

31 Print array KMESH part t.
This part contains the element numbers of the elements with highest dimensionality.
KMESH_KPROB must be equal to array KMESH.

32 Print array KMESH part u.
This part contains information about the spectral elements.
KMESH_KPROB must be equal to array KMESH.

33 Print array KMESH part v.
This part contains about element groups coupled to element groups.
KMESH_KPROB must be equal to array KMESH.

34 Print array KMESH part w.
This part contains information about real and integer properties.
KMESH_KPROB must be equal to array KMESH.

35 Print array KMESH part x information about obstacles.
This part contains.
KMESH_KPROB must be equal to array KMESH.

36 Print array KMESH part z.
This part contains information about changes in the mesh in case of free boundaries.
KMESH_KPROB must be equal to array KMESH.

37 Print array KMESH part y.
This part contains information for unstructured finite volumes.
KMESH_KPROB must be equal to array KMESH.

38 Print array KMESH part i.
This part contains the co-ordinates of the nodes.
KMESH_KPROB must be equal to array KMESH.

Input

ICHOICE must have a value.
The required arrays must have been filled.

Output

Print of information.

15.4 Printing the matrix (subroutine PRINMT)

Description

Subroutine to print the large matrix.

Since in general the large matrix is huge it is recommended to use this subroutine for small problems only. Furthermore if the matrix must be printed, it is advised to use a compact storage (METHOD>4) just for the printing, because this reduces the amount of output and furthermore produces an output that is more easy to read.

Heading

```
subroutine prinmt ( intmat, matr, kprob )
```

Parameters

INTEGER INTMAT(∗), MATR(∗), KPROB(∗)

INTMAT Standard SEPRAN array containing information about the storage of the matrices.
This array must have been filled before for example by a SEPRAN starting subroutine.

MATR Standard SEPRAN array containing information about the matrix. MATR must have been filled before for example by subroutine BUILD (7.2).

KPROB Standard SEPRAN array containing information about the problem description. KPROB must have been filled before for example by a SEPRAN start routine that reads the mesh.

Input

The arrays INTMAT, MATR and KPROB must have been filled.

Output

The large matrix is printed.
The sequence of the printing depends on the storage scheme.
For a symmetrical profile matrix, the matrix is printed row by row, where only the lower triangular elements within the profile are printed.
For a non-symmetrical profile matrix, per row not only the lower triangular elements are printed, but each row is followed immediately by the corresponding column in the upper triangle starting from the diagonal and going upwards.
In case of a compact storage only the non-zero elements are printed provided with row and column numbers. The output is self-explained.
Since the matrix is printed in the internal sequence of the degrees of freedom, it is recommended to print the new sequence with subroutine PRINI2 (15.3).

15.5 Printing special items related to the mesh (subroutine PRINTCURVES)

In this section we describe some special subroutines to print selected items from the mesh. The following subroutines are available:

PRINTCURVES (15.5.1]), prints the coordinates of the user points or the curves of the mesh, together with curve and node numbers.

15.5.1 Subroutine PRINTCURVES

Description

prints the curves or the user points of the mesh

Heading

```
subroutine printcurves ( ibuffr, buffer, kmesh, iprins )
```

Parameters

INTEGER IBUFFR(∗), KMESH(∗), IPRINS(10)

DOUBLE PRECISION BUFFER(∗)

IBUFFR Standard integer buffer array as stored in blank common. The reason to put this parameter in the parameter list is that in this way blank common may be avoided.

BUFFER Standard double precision buffer array. In fact this is exactly the same array as IBUFFR.

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

IPRINS Integer array to be filled by the user with the following information:

1 Indication if user points (1) or curves must be printed (2)

2 IINNER Information of how the curves must be printed

1 only the outer boundaries are printed

2 only the inner boundaries are printed

3 all boundaries are printed

3-10 Not yet used (Fill zeros for future developments).

Input

The arrays KMESH and IPRINS must have been filled.
The arrays IBUFFR and BUFFER must have been filled.

Output

Print of the coordinates of the user points or curves

16 Plot output

16.1 Introduction

In this section we shall describe some subroutines that may be used to plot results in a user written program. Most of the subroutines described in this chapter are part of the SEPRAN post processor SEPPOST. The plot routines create output in the way indicated by the environment file, depending on the plot device parameter posdev. See the Users Manual Section 1.5 for details. As a consequence the result of the plot routines may be a SEPRAN neutral plot file, a plot to the screen or alternatively a postscript file or a HPGL file.

In this chapter we deal with the following paragraphs:

16.2 defines some general definitions used in the plot subroutines. With the aid of this section it is possible to influence the layout of the plots. If none of the special actions in this section is used, a standard frame work and layout is produced.

16.3 treats the plot mesh subroutine PLOTMS. This subroutine replaces some previous mesh plot subroutines, which are also described.

16.4 deals with the general contour plot subroutine. This subroutine replaces some previous contour plot subroutines, which are also described.

16.5 describes the vector plot subroutine.

16.6 treats how to make function plots.

16.7 deals with subroutines to make a three-dimensional plot of a two-dimensional function.

16.8 is devoted to particle tracing.

16.9 deals with a special subroutine that is used to plot electric field lines.

16.10 describes a special subroutine that may be used to plot meshes that have been distorted due to a computed displacement vector.

16.11 describes special subroutines that may be used to plot curves and so on.

16.2 Some general definitions and help subroutines

In this section we shall describe some parameters, common blocks and subroutines that are common for all SEPRAN plot subroutines.

General parameters

The following parameters are common for the plot subroutines:

PLOTFM Length of the plot in centimeters.
PLOTFM is a double precision parameter, hence a real value must be submitted.
When PLOTFM < 0 the length of the plot in the x-direction is given by
- PLOTFM $\times$ dx, and in the y-direction by - PLOTFM $\times$ dy $\times$ YFACT, where dx is the maximal difference of x-co-ordinates in the region and dy the same for the y-co-ordinates.

YFACT Scale factor; all y-coordinates are multiplied by YFACT before plotting. YFACT $\neq 1$ should be used when the co-ordinates in x and y direction are of different scales, and hence the picture becomes too small.

General commons

Common block CPLOT is used by nearly all plot subroutines. It is initialized by the SEPRAN starting subroutines, but the user may change it for his own purposes.
Besides that common block CGRENS is used for plotting of axis.

CPLOT

Description

Common block CPLOT contains some parameters for the plot subroutines.

Declaration

```
      integer jmax, jmark, jframe, jtimes
      double precision xmin, xmax, ymin, ymax, zmin, zmax
      common /cplot/ xmin, xmax, ymin, ymax, zmin, zmax,
     +               jmax, jmark, jframe, jtimes
```

Parameters

xmin,xmax,ymin,ymax,zmin,zmax Minimum respectively maximum x, y and z-values of the region to be considered.
Hence only the points in the plotting area $xmin \leq x \leq xmax$, $ymin \leq y \leq ymax$, $zmin \leq z \leq zmax$ are plotted.

jmax Indication if the parameters xmin, xmax, ymin and ymax must be computed by the subroutine (0) or are given by the user (1).

jmark Flags, used to suppress drawing of scale-values. The following values are available:

0-99	Depends on the specific subroutine.
100	Suppress values along the x-axis.
1000	Same along left-hand y-axis.
10000	Same along right-hand y-axis (if selected).

jframe Parameter to indicate whether a frame work will be plotted as well as axis with scales. Possible values:

≥ 0	(standard) A plot frame is plotted without scales.
-1	No plot frame nor scales.
-2	A plot frame is plotted as well as axis with scales.
-3	No plot frame is plotted, the axis are plotted with scales.

The following texts are plotted in the frame-work depending on the value of JFRAME:

1 No text.

2 MESH.

3 FACTOR.

4 BOUNDARIES.

≥ 10 Submesh with sequence number JFRAME-10.

jtimes Parameter to indicate whether one or more plots are made in one picture. Possibilities:

0 Only one plot is made.

1 First plot in a sequence.

2 Next plot in a sequence.

3 Final plot in a sequence.

≥ 11 See jtimes-10, however, the plotting of axis and arrows in the standard plotting subroutines (except PLOTFN) is suspended. This possibility is meant for program SEPPOST. In this program the axis and arrows are plotted later on.

CGRENS

Description

Common block CGRENS contains information about the axis to be plotted.

Declaration

```
      integer ichoicex, ichoicey, isigx, isigy, nstepx, nstepy
      double precision axmin, axmax, aymin, aymax
      common /cgrens/ axmin, axmax, aymin, aymax, ichoicex, ichoicey,
     +                isigx, isigy, nstepx, nstepy
```

Parameters

axmin,axmax,aymin,aymax These parameters indicate the smallest respectively largest values to be plotted along the axis in x- and y-direction.

ichoicex Indication if axmin and axmax must be computed (-1) or not (Any other value).
When > 0 then numbers will be plotted in fixed-point format, else a floating point format will be used.

ichoicey Same for the y-axis.

isigx Number of significant digits (floating) or decimal places (fixed point) along x-axis.

isigy Same for the y-axis.

nstepx Number of scale values to be plotted along x-axis.

nstepy Same for the y-axis.

General subroutines

At this moment there is one general plotting subroutine (PLOTKD) that is called by each other plotting subroutine.

Subroutine PLOTKD

Description

Subroutine PLOTKD is a user written subroutine, that provides the user to plot extra information in his picture. Subroutine PLOTKD is only called when a frame-work is plotted.
PLOTKD is also called in the special case of JKADER = -4 in common block CPLOT. In that case the subroutine may be used to plot the frame work as well.

Call

```
CALL PLOTKD ( IDIREC, WID )
```

INTEGER IDIREC

REAL WID

IDIREC Parameter indicating in which direction the picture is plotted. When IDIREC = 1 the picture is plotted in the length direction of the paper (normal direction). When IDIREC = -1 the picture is plotted in the width direction of the paper (rotated direction). In that case all plots have been rotated with an angle of $-90°$ and translated with a factor WID.

WID Real variable giving the width of the plot in centimeters. The parameter WID must be used when IDIREC = -1.

Input

The parameters IDIREC and WID must have a value, initialized by the SEPRAN plot routine.

Output

The frame-work has the dimensions plotted in Figure 16.2.1. The user may plot texts in the

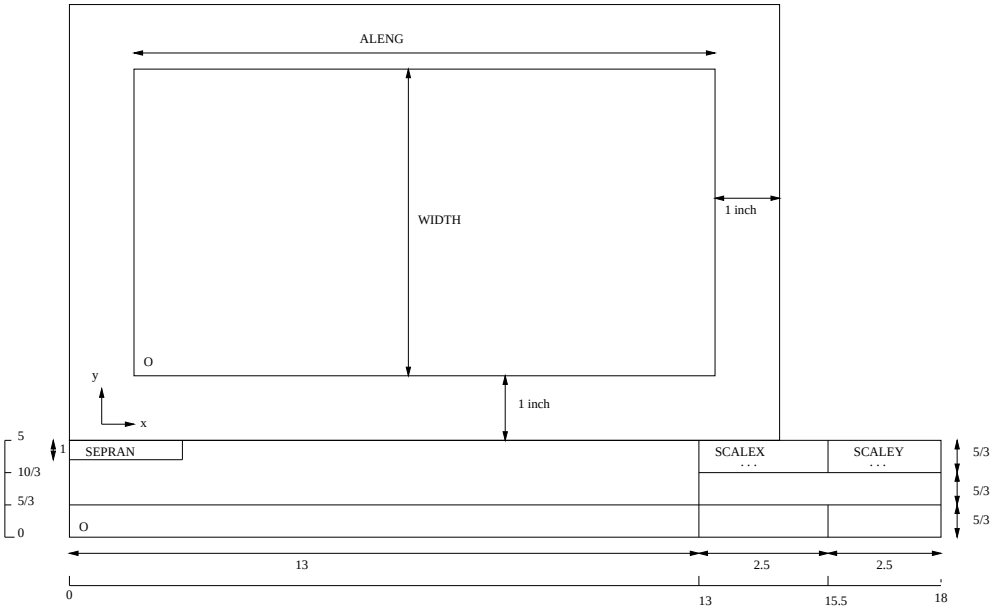


Figure 16.2.1: Frame-work, all measures in centimeters, with origin O

empty boxes; the dimensions are given in Figure 16.2.1.

The co-ordinates of the left under point O are (0,0) when IDIREC = 1 and (0,WID) when IDIREC = -1.
To plot texts and numbers the user must utilize machine dependent subroutines.
When JKADER = -4, the user must plot the frame work himself. At the end of the subroutine he must move the origin from the left under point to the point O in the plot. See Figure 16.2.1.

Example

Suppose the user wants to plot the text TIME = t in the frame-work block:
(13 , 5/3), (18 , 5/3), (18 , 10/3), (13 , 10/3) with t the actual time.

The time t is given in the user common block USTIME:
COMMON /USTIME/ t
Then the following subroutine has to be written:

```
      subroutine plotkd ( idirec, wid )
      implicit none
      integer idirec
      real wid
      double precision t
      common /ustime/ t
      real fpn

      if ( idirec .eq. -1 ) then

c         idirec = 1  normal direction
c         statements to fill the text time =
c         in this example the calcomp subroutines symbol and number are used

          fpn = t
          call symbol ( 13.5, 2.4, .1, 'time =', 0, 7 )
          call number ( 999. , 2.4, .1, fpn, 0, 2 )

      else

c         idirec = 1   rotated direction

          fpn = t
          call symbol ( 2.4, wid-13.5, .1, 'time =', -90., 7 )
          call number ( 2.4, 999., .1, fpn, -90., 2 )

      endif
      end
```

16.3 Making a mesh plot (subroutine PLOTMS)

In this section we shall describe the general SEPRAN mesh plot subroutine PLOTMS.
In previous versions of SEPRAN several mesh plotting subroutines were available:

PLOTM1 (16.3.2) Line plot of one-dimensional mesh.

PLOTM2 (16.3.3) Line plot of two-dimensional mesh.

PLOTM3 (16.3.4) Line plot of three-dimensional mesh with hidden line removal.

PLOT3M (16.3.5) Filled coloured polygon plot of three-dimensional mesh.

These subroutines are treated at the end of this section. Subroutine PLOTMS contains all the possibilities these old subroutines have. In fact at this moment PLOTMS actually calls each of these subroutines.

16.3.1 Subroutine PLOTMS

Description

General mesh plotting subroutine. This subroutine may be used for 1, 2 and 3-dimensional meshes. It is capable of making line plots (in 3D with hidden line removal) and of filled coloured polygon plots.

Call

```
CALL PLOTMS ( KMESH, IINPLT, RINPLT, ISKIP )
```

Parameters

INTEGER KMESH(∗), IINPLT(∗), ISKIP(∗)

DOUBLE PRECISION RINPLT(∗)

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

IINPLT General integer input array. With this array the user may make some choices of how he wants to plot the mesh. The length of the array depends on the number of entries that have been filled by the user. At present only 14 entries have been defined, but in the future this number may be extended. However, the construction is such that there is no need to adapt programs using PLOTMS in case of future releases unless the user wants to utilize the extra options then provided. The entries of IINPLT must be filled by the user as follows:

1: LAST Number of entries of IINPLT that are filled by the user. IINPLT(1) may be 0 or 1 in which case only default values are used. For all positions beyond position IINPLT(1), defaults are used. If a user fills an entry of IINPLT with a zero, also the default value is used.

2: PLOT_ELEM Indicates whether element numbers must be plotted in the centroid (1) or not (0). In three-dimensional meshes this option is neglected.
Default value: 0

3: PLOT_NODE Indicates whether nodal point numbers must be plotted (1) or not (0). In three-dimensional meshes this option is neglected.
Default value: 0

4: MARK_NODE Indicates whether nodal points must be marked with a star(1) or not (0). In three-dimensional meshes this option is neglected.
Default value: 0

5: CONTRACT Indicates whether elements must be contracted (1) or not (0).
Contraction only carried out for two-dimensional line plots. The contraction is defined as follows:
Let the centroid (x_c, y_c) be given by

$$x_c = \frac{\sum_{i=1}^{n_e} x_i}{n_e}, y_c = \frac{\sum_{i=1}^{n_e} y_i}{n_e}, \tag{16.3.1}$$

with (x_i, y_i) the co-ordinates of all points on the boundary of element e and n_e the number of boundary points. The distance $\mathbf{dx}_i = \mathbf{x}_i - \mathbf{x}_c$ is multiplied by 0.8 in order to get new points $(x_i^{'}, y_i^{'})$. Hence $(\mathbf{x}_i^{'} = \mathbf{x}_c + 0.8\mathbf{dx}_i$ These points are connected. In that way the new elements are strictly disjunct.
Default value: 0

6: PLOT_BOUN_DOUBLE Indicates whether the outer boundaries of a two-dimensional mesh must be plotted with a bold line (1) or not (0).

In a three-dimensional mesh that is plotted with filled coloured polygons the parameter indicates if the boundaries of the faces must be plotted (1) or not (0). These boundaries are plotted in the standard plotting colour of the display (usually white). In fact these boundaries clearly indicate the hidden line figure. On a black and white display the colour of the faces should be black. Combined with white boundaries, this results in a classical hidden-line picture.
Default value: 0

7: SKIP_GROUP Indicates whether element groups must be skipped (> 0) or not (0).
Default value: 0

8: REN_PLOT Indicates whether the original node numbers must be plotted (0) or the renumbered node numbers (1). This option makes only sense in combination with PLOT_NODE=1.
Default value: 0

9: COLOUR Indicates whether a filled coloured mesh plot (1) must be made or a line plot (0). This option can only be used in case of two- and three-dimensional elements.
Default value: 0

10: AXIS Indicates whether axis must be plotted (1) or not (0). In case of two-dimensional meshes the axis are provided with scales in three-dimensions always a small axis is plotted for the orientation.
Default value: 0

11: PROJECT Indicates the type of projection to be used in case of a three-dimensional plot. The following values are possible:

- 0 A perspective plot is made. The view point is explicitly given by the user in array RINPLT.
- 1 A parallel plot is made. The view point is explicitly given by the user in array RINPLT.
- 10 A perspective plot is made. The view point is computed automatically.
- 11 A parallel plot is made. The view point is computed automatically.

12: FAR_COLOUR Gives the colour sequence number of the element with the largest distance to the observer. Which colour actually is coupled to this number depends completely on the plot package and the local installation.
This option is only used if COLOUR=1.
Default value: 1

13: NEAR_COLOUR Gives the colour sequence number of the element with the smallest distance to the observer. Which colour actually is coupled to this number depends completely on the plot package and the local installation.
On a black and white terminal only colour 0 may be used, on for example a 256-colour display the colours to be used may vary from 0 to 256. The colour of a specific face is computed by linear interpolation between nearest and farthest colour.
This option is only used if COLOUR=1.
Default value: 16

14: ORIENTATION Defines the orientation of the axis. This option is only used in three dimensions. The following values are possible:

1 The standard x-y-z co-ordinate system is used

2 The z-x-y co-ordinate system is used

3 The y-z-x co-ordinate system is used

Default value: 1

15: NUMROTATIONS Number of rotations to be performed for a three-dimensional mesh. The rotations are such that the complete 360 degrees are subdivided into equal parts.
Default value: 1

16: IVOLUMES If larger than 0, volumes must be skipped, otherwise (0) all volumes must be plotted (3d only). If volumes must be skipped, array ISKIP must be used to indicate which volumes must be plotted.
Default value: 0

17: IROWPLOT If 1 elements in a 3D mesh are plotted per layer. This may be used to check the quality of internal elements.
Default value: 0

18: IDEFSTART is only used if IROWPLOT=1. It defines how the starting set of nodes must be computed.
Possible values:

1 Start with set of nodes with largest distance to view point.

2 Start with set of surfaces indicated in positions 19 and 20

Default value: 1

19: IFIRST is only used if IDEFSTART=2. It defines the first surface number.
Default value: 1

20: ILAST is only used if IDEFSTART=2. It defines the last surface number.
Default value: IFIRST

RINPLT General real input array to be filled by the user. The length depends on the type of mesh to be plotted as well as the array IINPLT. Contents:

1: PLOTFM Length of the plot in centimeters.

2: YFACT Scale factor; all y-coordinates are multiplied by YFACT before plotting.
YFACT $\neq 1$ should be used when the co-ordinates in x and y direction are of different scales, and hence the picture becomes too small.

3-5: EYEPOINT co-ordinates of point where the observer of the mesh is positioned. The plot strongly depends on the position of the observer.
These positions are only used in case of a three-dimensional mesh when
PROJECT < 10.

ISKIP In this array it is stored which element groups must be skipped if SKIP_GROUP > 0.
ISKIP(ielgrp) = 0 means skip element group ielgrp, ISKIP(ielgrp) = 1 means do not skip.

Input

The arrays IINPLT and RINPLT must have been filled by the user.
Array KMESH must have been filled.
Depending on the parameter SKIP_GROUP array ISKIP must have been filled.

Output

Plot of the mesh.

Subroutines used in previous versions of SEPRAN

As indicated in the introduction of this section in previous versions of SEPRAN the subroutines PLOTM1, PLOTM2, PLOTM3 and PLOT3M have been used for the plotting of meshes. Here we shall give a short description of these subroutines. Although not longer recommended they still may be used in future versions of SEPRAN.

16.3.2 Subroutine PLOTM1

Description

A one-dimensional mesh (or a part of it) will be plotted.

Call

```
CALL PLOTM1 ( ICHOICE, KMESH, IUSER, PLOTFM )
```

Parameters

INTEGER ICHOICE, KMESH(∗), IUSER(∗)

DOUBLE PRECISION PLOTFM

ICHOICE Parameter to distinguish between the following possibilities:

0 Standard method, the whole mesh is plotted.

1 Only a part of the mesh is plotted, depending on the values in array IUSER.
The positions 6 to NELGRP + 5 of array IUSER must contain the following information:
IUSER(IELGRP + 5) = 0 implies that the elements with standard element group number IELGRP must be skipped, whereas
IUSER(IELGRP + 5) = 1 implies that these elements are plotted.

2 The complete mesh will be plotted, however, each element is contracted in the following way: let the centroid x_c be given by: $(x_1 + x_n)/2$, with n the number of nodes in the element. The distance x_i to x_c is multiplied by 0.8 in order to get new points $(x_i^{'}$. These points are connected. In that way, the new elements are strictly disjunct.

3 Only a part of the mesh is plotted (see ICHOICE = 1).
The elements are contracted (see ICHOICE = 2).

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

IUSER Array IUSER will only be used when ICHOICE = 1 or 3.

$$\text{length} \begin{cases} 0 & \text{when ICHOICE } = 0, 2 \\ 5 + \text{ NELGRP} & \text{when ICHOICE } = 1, 3 \end{cases}$$

PLOTFM Length of plot in centimeters, see Section 16.2.

Parameters in common block CPLOT

Besides the standard input parameters also the parameters in common block CPLOT are used as described in Section 16.2. Extra possibilities are:

JMARK Parameter consisting of three parts i1, i2 and i3 according to JMARK = i1 + i2 + i3

These parameters have the following meaning:

i1

0 Each nodal point in the plotting region is marked with a star, and its nodal point number.

1 Each nodal point in the plotting region is marked with a star. Nodal point numbers are **not** plotted.

2 Nodal points are not marked, nor are nodal point numbers plotted.

i2

0 Element numbers are plotted in the centroid of the elements.

3 Element numbers are not plotted.

i3

0 The original nodal point numbering is plotted.
10 The renumbered nodal point numbering is plotted.

Input

Array KMESH must have been filled.
ICHOICE, PLOTFM and YFACT must have a value.
NDIM must have the value 1.
When ICHOICE = 1 or 3, array IUSER must be filled.
The parameters in common block CPLOT must have a value.

Output

Plot of the mesh.

16.3.3 Subroutine PLOTM2

Description

A two-dimensional mesh (or a part of it) will be plotted.

Call

```
CALL PLOTM2 ( ICHOICE, KMESH, IUSER, PLOTFM, YFACT )
```

Parameters

INTEGER ICHOICE, KMESH(∗), IUSER(∗)

DOUBLE PRECISION PLOTFM, YFACT

ICHOICE Parameter to distinguish between the following possibilities:

0 Standard method, the whole mesh is plotted.

1 Only a part of the mesh is plotted, depending on the values in array IUSER.
The positions 6 to NELGRP + 5 of array IUSER must contain the following information:
IUSER(IELGRP + 5) = 0 implies that the elements with standard element group number IELGRP must be skipped, whereas
IUSER(IELGRP + 5) = 1 implies that these elements are plotted.

2 The complete mesh will be plotted, however, each element is contracted in the following way: let the centroid (x_c, y_c) be given by:

$$x_c = \frac{\left(\sum_{i=1}^{n_e} x_i\right)}{n_e} \qquad y_c = \frac{\left(\sum_{i=1}^{n_e} y_i\right)}{n_e}$$

with (x_i, y_i) the co-ordinates of all points on the boundary of element e and n_e the number of boundary points. The distance (x_i, y_i) to (x_c, y_c) is multiplied by 0.8 in order to get new points $(x_i^{'}, y_i^{'})$. These points are connected. In that way, the new elements are strictly disjunct.

3 Only a part of the mesh is plotted (see ICHOICE = 1).
The elements are contracted (see ICHOICE = 2).

≥ 10 See ICHOICE - 10
The outer boundaries are plotted with double lines.

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

IUSER Array IUSER will only be used when ICHOICE = 1 or 3.

$$\text{length} \begin{cases} 0 & \text{when ICHOICE } = 0, 2 \\ 5 + \text{ NELGRP} & \text{when ICHOICE } = 1, 3 \end{cases}$$

PLOTFM Length of plot in centimeters, see Section 16.2.

YFACT Scale factor, see Section 16.2.

Parameters in common block CPLOT

Besides the standard input parameters also the parameters in common block CPLOT are used as described in Section 16.2. Extra possibilities are:

JMARK Parameter consisting of three parts i1, i2 and i3 according to JMARK = i1 + i2 + i3

These parameters have the following meaning:

i1

0 Each nodal point in the plotting region is marked with a star, and its nodal point number.

1 Each nodal point in the plotting region is marked with a star. Nodal point numbers are **not** plotted.

2 Nodal points are not marked, nor are nodal point numbers plotted.

i2

0 Element numbers are plotted in the centroid of the elements.

3 Element numbers are not plotted.

i3

0 The original nodal point numbering is plotted.

10 The renumbered nodal point numbering is plotted.

Input

Array KMESH must have been filled.
ICHOICE, PLOTFM and YFACT must have a value.
NDIM must have the value 2.
When ICHOICE = 1 or 3, array IUSER must be filled.
The parameters in common block CPLOT must have a value.

Output

Plot of the mesh.

16.3.4 Subroutine PLOTM3

Description

A three-dimensional mesh will be plotted.
Lines that should be not visible from the position of the user, are not plotted (hidden-line method).

Call

```
CALL PLOTM3( ICHOICE, KMESH, IUSER, PLOTFM, YFACT, VIEWPN )
```

Parameters

INTEGER ICHOICE, KMESH(∗), IUSER(∗)

DOUBLE PRECISION PLOTFM, YFACT, VIEWPN(3)

ICHOICE Parameter consisting of three parts IPERSP, IVIEW and IORIEN according to:
ICHOICE = IPERSP + 10*IVIEW + 100*IORIEN

These parameters have the following meaning:

IPERSP

0 perspective projection plot
1 parallel projection plot

IVIEW

0 view point given in array VIEWPN
1 view point computed automatic

IORIEN

0 Standard orientation of axis (x-y-z)
1 orientation of axis is (z-x-y)
2 orientation of axis is (y-z-x)

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

IUSER Not used.

PLOTFM Length of plot in centimeters, see Section 16.2.

YFACT Scale factor, see Section 16.2.

VIEWPN In this array the eye point must be stored, see PLOTMS (16.3.1).

Input

The arrays KMESH and VIEWPN must have been filled.
ICHOICE, PLOTFM and YFACT must have a value.
NDIM must have the value 3.
The parameters in common block CPLOT must have a value.

Output

Hidden line plot of the mesh.

16.3.5 Subroutine PLOT3M

Description

PLOT3M makes a plot of a three-dimensional mesh by filling (coloured) polygons. The subroutine starts to plot at the farthest faces and successively plots all faces closer to the observer. The closest surface is plotted last. In this way a hidden line plot with coloured faces arises automatically. The colour indicates the distance to the observer. This procedure can **only** be applied on a non-static display. If a black and white display is used, the final plot is also a standard hidden line plot. This way of plotting is much faster than by using PLOT3D (16.7). However, a plot with a pen plotter can not be made. In order to get a hard copy, one needs a hard copy unit which copies the prints of the screen.

Remark: in fact this subroutine creates a hidden line plot on a screen in a relative cheap way.

Call

```
CALL PLOT3M ( KMESH, IPLOTS, RPLOTS )
```

Parameters

INTEGER KMESH(∗), IPLOTS(∗)

DOUBLE PRECISION RPLOTS(∗)

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

IPLOTS General integer input array. With this array the user may make some choices of how he wants to plot the mesh. The length of the array depends on the number of entries that have been filled by the user. At present only 14 entries have been defined, but in the future this number may be extended. The entries of IPLOTS must be filled by the user as follows:

1: LAST Number of entries of IPLOTS that are filled by the user. IPLOTS(1) may be 0 or 1 in which case only default values are used. For all positions beyond position IPLOTS(1), defaults are used. If a user fills an entry of IPLOTS with a zero, also the default value is used.

2: PROJECT_METHOD Only value 1 is available.

3: FAR_COLOUR See PLOTMS (16.3.1).

4: NEAR_COLOUR See PLOTMS (16.3.1).

5: PLOT_BOUN_DOUBLE See PLOTMS (16.3.1).

6: PROJECT Indication if parallel projection must be used (0) or perspective projection (1).

7: ISKIP Indication if element groups must be excluded (1) or not (0). If ISKIP = 1, positions 15 to 14+NELGRP of array IPLOTS should be filled.

8: NUMROTATIONS See subroutine PLOTMS, IINPLT(15)

9: IVOLUMES See subroutine PLOTMS, IINPLT(16)

10: IROWPLOT See subroutine PLOTMS, IINPLT(17)

11: IDEFSTART See subroutine PLOTMS, IINPLT(18)

12: IFIRST See subroutine PLOTMS, IINPLT(19)

13: ILAST See subroutine PLOTMS, IINPLT(20)

14 Not used

15,...,14+NELGRP These positions are only used if ISKIP = 1.
In that case IPLOTS(14+*ielgrp*) indicates if element group *ielgrp* must be skipped (0) or must be taken into account (1).

RPLOTS See subroutine PLOTMS (16.3.1) array RINPLT.

Input

The arrays KMESH, IPLOTS and RPLOTS must have been filled.
NDIM must have the value 3.
The parameters in common block CPLOT must have a value.

Output

Coloured hidden-line plot on a display.

16.4 Making a contour plot (subroutine PLOTCN)

In this section we shall describe the general SEPRAN contour plot subroutine PLOTCN.
In previous versions of SEPRAN several mesh plotting subroutines were available:

PLOTC1 (16.4.2) Plot contour lines in R^2.

PLOTC2 (16.4.3) Coloured contour levels in R^2.

PLOTC3 (16.4.4) Compute and plot all contour lines for a given level in a mesh in R^2. The contour lines are stored in an output array.

CONT3C (16.4.5) Contour plot of function on 3D surface using filled coloured polygons.

CONT3D (16.4.6) Contour plot of function on 3D surface.

These subroutines are treated at the end of this section. Subroutine PLOTCN (16.4.1) contains all the possibilities these old subroutines have.

16.4.1 Subroutine PLOTCN

Description

General SEPRAN contour plotting subroutine.
Both 2D and 3D contour lines or coloured contour levels may be plotted

Call

```
CALL PLOTCN ( KMESH, KPROB, ISOL, IPLOTS, RPLOTS, CONTLN,
              INPUT, TEXT, NCOORC, COORCN )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), ISOL(5), IPLOTS(∗), INPUT(∗), NCOORC(∗)

DOUBLE PRECISION RPLOTS(∗), CONTLN(∗), COORCN(∗)

CHARACTER ∗ **(**∗**)** TEXT(∗)

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

KPROB Standard SEPRAN array, containing information of the problem definition.

ISOL Standard SEPRAN array of length 5 containing information concerning the solution vector or a vector of special structure from which the contours must be computed and plotted.

IPLOTS Integer control array in which the user defines which choices he wants for this subroutine IPLOTS must be filled by the user in the following way:

1 Defines the highest entry number of the array that has been filled by the user. This means that the user must fill all positions 1 to IPLOTS(1) in array IPLOTS. For all other positions default values are used. Hence if IPLOTS(1) = 0 or 1 only defaults are used.
Remark: if IPLOTS(i)=0, also for position i the default is taken.

2 (CHOICE) defines the type of contour lines or levels required. Possible values for CHOICE are:

1 Contour lines will be drawn. These lines are provided with a numbered label.
In the case of a 3D grid, contour lines are plotted on the outer surface only.

2 Contour levels will be coloured, i.e. the interior space between contour levels as well as the region smaller than the smallest contour level and the region larger than the largest contour level is filled with a specific colour for that level.

3 See 1, however, not only are the contour levels plotted, but also the co-ordinates of these contour lines are outputed in the arrays COORCN and NCOORC.
This possibility is only available in 2D.

Default value: 1.

3 (NUMBER) the NUMBER$^{\text{th}}$ physical unknown in each node is considered When this unknown does not appear in a nodal point, the point is skipped When array ISOL corresponds to a complex array, NUMBER=2×i-1 corresponds to the real part of unknown i and NUMBER=2×i to the imaginary part of unknown i.
Default value: 1.

4 (NCNTLN) Parameter indicating the number of contour levels to be used. Possible values of ncntln are:

> 0 There are ncntln contour levels that must be stored in array contln from position 6

0 The program computes the contour levels by dividing the range of the function between maximum and minimum value into 10 parts in the case of CHOICE = 1 or 3 and in 20 parts if CHOICE=2

$< 0, > -1000$ The program computes the contour levels by dividing the range of the function between maximum and minimum value into |NCNTLN| - 1 parts. The values are chosen such that rounded values of the contour levels appear

< -1000 The program computes the contour levels by dividing the range of the function between maximum and minimum value into |NCNTLN + 1000| - 1 parts.

Default value: 0.

5 (JINDCOL) Parameter indicating the type of colouring to be used for the contour lines or levels.
Possible values:

0 The standard colours are used, i.e. standard colour contour lines in the case of CHOICE=1 or CHOICE=3 and colour numbers 1 to number_of_contour_levels in the case of CHOICE=2.

1 The user defines the colour numbers explicitly through array INDCOL. This possibility is moment only available for CHOICE=2.

2 The user defines the minimum colour number explicitly through array INDCOL. This possibility is only available for CHOICE=2.

3 The user defines the minimum and maximum colour number explicitly through array INDCOL. This possibility is only available for CHOICE=2.

> 0 The visible parts of the contour lines get the colour sequence number JINDCOL CHOICE=1 or 3.

Default value: 0.

6 (PROJECT) choice parameter for type of projection (3D only)

0 parallel projection plot (view point given by the user in RPLOTS(4-6)).

1 perspective projection plot (view point given by the user in RPLOTS(4-6)).

10 parallel projection plot (View point automatic).

11 perspective projection plot (View point automatic).

Default value: 10.

7 (JSMOOTH) choice parameter for type of smoothing to be used for the contour lines. At this moment JSMOOTH is only implemented for CHOICE=1 or 3 in 2-dimensional grids. Possible values:

0 no smoothing

1 shuman smoothing

2-5 all spline smoothing : the higher the value the more smoothing and the less interpolation of the values.

Default value: 0.

8 (IDOUBLE) indication whether the outer boundary is plotted as a single line (0) or as double line (1) (2D only).
Default value: 0.

9 (INNER) indication whether inner boundaries must be plotted (1) or not (0). (2D only)
Default value: 0.

10 (PLOT_CONTOUR) indication if the boundaries of the contour levels must be plotted (> 0) or not (0). The value of PLOT_CONTOUR defines the colour. This possibility is only available in the case CHOICE=2.
Default value: 0.

11 (TYPE_LEGENDA) defines the type of legends to be plotted.
Possible values:

1 No legends

2 Standard legends; the text is plotted at the right side of the picture.

3 Horizontal legends; the text plotted below the picture. Only minimum and maximum values are given.
Only available for CHOICE=2.

Default value: 2.

12 (TYPE_SCALES) defines whether the scaling factors are plotted (1) or not (2).
Default value: 1.

13 (ISKIP) indicates if element groups must be excluded (>0) or not (0).
If ISKIP>0, the value of ISKIP defines the first position in INPUT where information about the element groups to be excluded is stored.
Default value: 0.

14 IDEFFM Indicates if default sizes of the plot are used (IDEFFM=0) or that the user gives these values in RPLOTS(1) and RPLOTS(2) (IDEFFM=1).
If IDEFFM=2 the user must also fill RPLOTS(3) with a pre multiplication factor for the solution The effect of the multiplication factor is that the contour levels are plotted for the scaled solution, the solution itself, however, is not influenced If the user gives the contour levels explicitly these contour levels must correspond to the scaled vector.
Default value: 0.

15 (IMESH) indicates if the mesh must be plotted in the picture (> 0) or not (0).
If IMESH>0, the value of IMESH defines the colour sequence number with which the mesh must be plotted. In 3D only the visible parts of the mesh are plotted with colour IMESH.
If in 3D CHOICE=2 is used, plotting of the mesh means plotting of triangles. In the case of quadrilaterals or quadratic elements this may mean that the mesh is subdivided into smaller triangles.
Default value: 0.

16 (INVIS_MESH) indicates if the invisible part of the mesh must be plotted in the picture (>0) or not (0) If INVIS_MESH>0, the value of INVIS_MESH defines the colour sequence number with which the invisible parts of the mesh must be plotted. (CHOICE=1 and 3D only)
Default value: 0.

17 (INVIS_CONT) indicates if the invisible part of the contour lines must be plotted in the picture (>0) or not (0). If INVIS_CONT>0, the value of INVIS_CONT defines the colour sequence number with which the invisible parts of the contours must be plotted. (CHOICE=1 and 3D only).
Default value: 0.

18 (BACK_COLOUR) indicates if back ground lines must plotted in the picture (>0) or not (0).
If BACK_COLOUR>0, the value of BACK_COLOUR defines the colour sequence number with which the back ground lines must be plotted. (CHOICE=1 and 3D only).
Default value: 0.

19 (CURV_COLOUR) indicates if curves must be plotted in the picture (>0) or not (0).
If CURV_COLOUR>0 and < 100, the value of CURV_COLOUR defines the colour sequence number with which the visible part of the curves must be plotted.
If CURV_COLOUR>100 also the invisible part of the curves is plotted, the value of CURV_COLOUR-100 defines the colour sequence number with which the curves must be plotted.
This possibility is only available for 3D.

20 (ITEXT) indicates if text must be plotted in the picture.
Possible values:

- 0 No text at all.
- 1 Extra text at bottom.
- 2 Extra text at bottom and top.

Default value: 1.

21 (ICOMPT) indicates how the contour levels must be computed from NCNTLN in the case NCNTLN ≤ 0

- 0 The minimum and maximum level are computed.
- 1 The minimum level is given in contln(6), the maximum level must be computed.
- 2 The maximum level is given in contln(7), the minimum level must be computed.
- 3 The minimum level is given in contln(6), the maximum level in contln(7).

Default value: 0.

22 (IORIENT) indicates the orientation of the base vectors.
Possible values:

- 1 x - y - z system (also default).
- 2 z - x - y system.
- 3 y - z - x system.

RPLOTS Double precision input array that must be filled as follows:

- 1 plotfm (length of the plot, see Section 16.2)
- 2 yfact (scaling factor, see Section 16.2)
- 3 scale factor. This parameter is only used if IDEFFM=2.
- 4-6 eye-point (Only if PROJECT>9).

CONTLN User array of length ncntln + 5 (if used) in which the user is supposed to have stored the contour levels from position 6.

INPUT This array is only used when JINDCOL (IPLOTS(5)) >0 and or ISKIP>0 In fact this array consists of three parts:

Part 1 are the first 5 positions. They have to be filled by the user in the following way:

- 1 Declared length of array input.
- 2-5 0

Part 2 contains information about the colours to be used. Part 2 starts at position 6 and the length depends on JINDCOL.
If JINDCOL=1, it must be filled by the user with the colours of the various contour levels in increasing sequence. The length is equal to the number of contour levels.

If JINDCOL=2 it has length 1.
In INDCOL(6) the user must store the minimum colour required.
If JINDCOL=3 it is has length 2.
The minimum colour must be stored in INDCOL(6) the maximum one in INDCOL(7).
If the range of the colours is less than the available range of colours the colour numbering is repeated.

Part 3 contains information about the element groups to be skipped. Part 3 is only used if ISKIP>0. The value of ISKIP defines the first position in INPUT with information about the element groups to be skipped. Part 3 must be filled as follows:
If INPUT(ISKIP-1+i)=0, element group i must be skipped.
If INPUT(ISKIP-1+i)=1, element group i must be taken into account.

TEXT Text array containing the texts to be plotted.

1 Text at bottom of the picture.

2 Text at top of the picture.

NCOORC This array contains at output for every contour the following information:

1 Must be filled by the user with the length of the array, because it is a variable length array.

2-5 These positions are not used.

next positions:

For any contour level $n + 1$ positions are used , where n is the number of parts the contour levels contain. A part is a continuous contour line. The contour levels are stored sequentially, starting with contour level 1.

1 contains the number n for the contour level ($n \geq 1$).

2 - n+1 contain of each part the number of points in the contour level part of which the co-ordinates are stored in array COORCN.
If the contour part is a closed contour, then minus the number of points on the contour part is stored.

So information of contour level 1 is stored in NCOORC from position 6 to position 6+n_1 etc.

COORCN At output this array contains the co-ordinates of the points on the contours in the sequence as given in NCOORC. These co-ordinates are stored from position 6 in the sequence (x_1, y_1), (x_2, y_2) etc. COORCN is a variable length array, hence the first position must be filled by the user.
COORCN is only filled if CHOICE (IPLOTS(2)) = 3.

Input

Some of the arrays IPLOTS, RPLOTS, CONTLN, INPUT and TEXT must have been filled by the user.
The arrays KMESH, KPROB, ISOL must have been filled.
Depending on the parameter SKIP_GROUP array ISKIP must have been filled.

Output

Contour plot.
Depending on IPLOTS also the arrays COORCN and NCOORC may have been filled.

Subroutines used in previous versions of SEPRAN

As indicated in the introduction of this section in previous versions of SEPRAN the subroutines

PLOTC1, PLOTC2, PLOTC3, CONT3C and CONT3D have been used for the plotting of contours. Here we shall give a short description of these subroutines. Although not longer recommended they still may be used in future versions of SEPRAN.

16.4.2 Subroutine PLOTC1

Description

A number of contour lines (lines with equal function value) of the solution will be plotted. (In two dimensions only).

Call

```
CALL PLOTC1 ( NUMBER, KMESH, KPROB, ISOL, CONTLN, NCNTLN,
              PLOTFM, YFACT, JSMOOTH )
```

Parameters

INTEGER NUMBER, KMESH(∗), KPROB(∗), ISOL(5), NCNTLN, JSMOOTH

DOUBLE PRECISION CONTLN(∗), PLOTFM, YFACT

NUMBER The NUMBER$^{\text{th}}$ unknown in each nodal point in array ISOL will be selected. When this unknown does not appear in a nodal point, this point is skipped.
When array ISOL corresponds to a complex vector then NUMBER = 1 corresponds to the real part of the first unknown in each nodal point and NUMBER = 2 to the imaginary part. In the same way NUMBER = 3 corresponds to the real part of the second unknown etc. Hence when the real part of the i^{th} unknown is required, NUMBER should have the value 2 × i - 1, for the imaginary part NUMBER should be equal to 2 × i.

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

KPROB Standard SEPRAN array, containing information of the problem definition.

ISOL Standard SEPRAN array of length 5 containing information concerning the solution vector or a vector of special structure from which the contours must be computed and plotted.

CONTLN Array CONTLN is only used when NCNTLN > 0.
In that case it is a variable length array of length NCNTLN + 5, hence position 1 must have a value.
The value of a contour level i must be filled in CONTLN(i+5) (i=1(1)NCNTLN).

NCNTLN The value of $|NCNTLN|$ gives the number of contour lines to be plotted. At most 1000 contours may be plotted.
When NCNTLN > 0 the values of the contour levels must be stored in array CONTLN from position 6.
When NCNTLN = 0, the program computes the contour values by dividing the range of the function between maximum and minimum value into 10 parts. For example when min=0 and max=1, then the contour lines 0, 0.1 ,0.2,. . ., 1 are plotted. The minimum and maximum values are also plotted. Array CONTLN is not used in this case.
When NCNTLN < 0 the program computes the contour values by dividing the range of the function between minimum and maximum value, into $|NCNTLN|$ - 1 parts. The values are chosen such that rounded values of the contour levels appear. For example 0, 0.1, 0.2, ... , 1.0 instead of 0, 0.095, 0.19, ... , 0.95 when the range is 0 to 0.95. Array CONTLN is not used in this case.
When NCNTLN < -1000 the program computes the contour values by dividing the range of the function between minimum and maximum value, into
$|NCNTLN + 1000|$ - 1 parts. Array CONTLN is not used in this case.

PLOTFM Length of plot in centimeters. Double precision variable, hence a double precision real must be submitted in this place.

YFACT Scale factor; all y-co-ordinates are multiplied by this factor before plotting the boundaries. Double precision real variable, hence a double precision real must be submitted.
YFACT $\neq 1$ is used when the co-ordinates in x and y-direction are of different scales, and hence the picture becomes too small.

JSMOOTH Depending on the value of JSMOOTH the contour lines are smoothed.
Possibilities:

0: No smoothing takes place. The result is a piece-wise linear contour.

1: A kind of mean value between 3 succeeding values of the contour is computed. This so-called Shuman filter smoothes some of the possible wiggles.

2-5: The contour is approximated by a smooth spline. This gives beautiful contours, extremely suited for publication and presentation of contours. However, the computation of these contours is more expensive and it may be possible that significant features of the computations may be smoothed out. The smoothness of the spline is increased for increasing values of JSMOOTH, hence JSMOOTH = 5 gives the maximal smoothness available.

10≤JSMOOTH≤15: See JSMOOTH-10; the outer boundaries are plotted with double lines.

100≤JSMOOTH≤115: See JSMOOTH-100; the inner boundaries are plotted also.

Input

The arrays KMESH, KPROB, ISOL, KBNDPT and when NCNTLN > 0, CONTLN must be filled. The parameters NUMBER, NCNTLN, PLOTFM, YFACT and JSMOOTH must have a value.

Output

Plot of the contour lines.

16.4.3 Subroutine PLOTC2

Description

The interior space between contour levels as well as the region smaller than the smallest and the region larger than the largest contour level is filled with a specific colour for that level. Hence from the colour of the plot it can be seen in what the range the function value at a point is situated.

Call

```
CALL PLOTC2 ( NUMBER, KMESH, KPROB, ISOL, CONTLN, NCNTLN, INDCOL,
              PLOTFM, YFACT, ICHOICE )
```

Parameters

INTEGER NUMBER, KMESH(∗), KPROB(∗), ISOL(5), NCNTLN, INDCOL(∗), ICHOICE

DOUBLE PRECISION CONTLN(∗), PLOTFM, YFACT

NUMBER, KMESH, KPROB, ISOL, CONTLN, NCNTLN, PLOTFM, YFACT See subroutine PLOTC1 (16.4.2).

INDCOL Array INDCOL is only used if JINDCOL=1 (see ICHOICE).
In that case INDCOL is a variable length array of length NCNTLN+6, hence position 1 must have a value.
In INDCOL(i+5) a number must be stored indicating the type of colour to be used for the region:

i=1: function value < contour level 1.

$1 < i \leq$last level: contour level i-1 < function value < contour level i.

$i >$last level: function value > last contour level.

When INDCOL is not used, the colours with colour index 1 to last level + 1 are used. Inform at your local installation personnel which colours correspond to these colour indices.

ICHOICE Choice parameter with the following meaning:
ICHOICE = JINDCOL + 10 × IDOUBL + 100 × IINNER, with

JINDCOL Indication if array INDCOL is filled by the user (JINDCOL=1) or that the colours 1 to number of contour levels plus one are used (JINDCOL=0).

IDOUBL Indication if the outer boundaries are plotted normally (IDOUBL=0), or with double lines (IDOUBL=1)

IINNER Indication if the inner boundaries are plotted (IINNER=1) or not (IINNER=0)

Input

The arrays KMESH, KPROB, ISOL and when NCNTLN > 0, CONTLN must be filled.
When JINDCOL=1, array INDCOL must be filled.
The parameters NUMBER, NCNTLN, PLOTFM, YFACT and ICHOICE must have a value.

Output

Colour plot of the contour levels.

16.4.4 Subroutine PLOTC3

Description

Subroutine PLOTC3 is exactly identical to PLOTC1 (16.4.2), however, PLOTC3 contains two extra parameters giving the user the possibility to store the contours himself.

Call

```
CALL PLOTC3 ( NUMBER, KMESH, KPROB, ISOL, CONTLN, NCNTLN,
              PLOTFM, YFACT, JSMOOTH, NCOORC, COORCN )
```

Parameters

INTEGER NUMBER, KMESH(∗), KPROB(∗), ISOL(5), NCNTLN, JSMOOTH, NCOORC(∗), COORCN(∗)

DOUBLE PRECISION CONTLN(∗), PLOTFM, YFACT

NUMBER, KMESH, KPROB, ISOL, CONTLN, NCNTLN, PLOTFM, YFACT, JSMOOTH See subroutine PLOTC1 (16.4.2).

NCOORC, COORCN See subroutine PLOTCN (16.4.1).

Input

The arrays KMESH, KPROB, ISOL, KBNDPT and when NCNTLN > 0, CONTLN must be filled.
The parameters NUMBER, NCNTLN, PLOTFM, YFACT and JSMOOTH must have a value.
The first positions of COORCN and NCOORC must have been filled.

Output

Plot of the contour lines.
The arrays COORCN and NCOORC have been filled.

16.4.5 Subroutine CONT3C

Description

Subroutine CONT3C may be used to make a coloured level plot of a function defined on a three-dimensional region. The surfaces between two contour levels are filled with one colour.

This subroutine plots filled triangles from the back to the front. Hence the filled faces at the backside may be overwritten by those at the front. As a consequence this subroutine may only be applied on a display not with a plotter.

Call

```
CALL CONT3C ( KMESH, KPROB, ISOL, IPLOTS, RPLOTS )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), ISOL(5), IPLOTS(∗)

DOUBLE PRECISION RPLOTS(∗)

KMESH, KPROB, ISOL See subroutine PLOTC1 (16.4.2).

IPLOTS Integer array of length 6+NELGRP, where NELGRP denotes the number of element groups.
IPLOTS must be filled by the user in the following way:

1: NUMBER; The NUMBER$^{\text{th}}$ unknown in each nodal point in array corresponding to ISOL will be selected. When this unknown does not appear in a nodal point, this point is skipped.

2: Choice parameter for the projection method to be used. Possible values:

1 Perspective projection.

2 Parallel projection.

3: Sequence number of the farthest colour to be used.

4: Sequence number of the nearest colour to be used. The colour numbers that may be used depend on the installation of your plotting package and the type of display to be used. On a black and white terminal only colour 0 may be used, on for example a 256-colour display the colours to be used may vary from 0 to 256. The colour of a specific face is computed by linear interpolation between nearest and farthest colour.

5: Indication if the boundaries of the contour levels must be plotted (1) or not (0). These boundaries are plotted in the standard plotting colour of the display (usually white). In fact these boundaries clearly indicate the hidden line contour plot. On a black and white display the colour of the faces should be black. Combined with white boundaries, this results in a classical hidden-line contour plot.

6: ISKIP; Indication if element groups must be excluded (1) or not (0). If ISKIP = 1, positions 7 to 6+NELGRP must have been filled.

7-6+NELGRP: These positions are only used if ISKIP = 1.
In that case IPLOTS(6+*ielgrp*) indicates if element group *ielgrp* must be skipped (0) or must be taken into account (1).

RPLOTS Double precision array of length 5.
RPLOTS must be filled by the user in the following way:

1 PLOTFM; Length of plot in centimeters.

2 YFACT; Scale factor; all function values are multiplied by this factor before plotting.

3-5 Co-ordinates of the eye-point, i.e. the point where the observer is positioned. The co-ordinates must be stored in the sequence x, y, z.

Input

The arrays KMESH, KPROB, ISOL, IPLOTS and RPLOTS must have been filled.

Output

Coloured contour-level plot consisting of filled polygons.

16.4.6 Subroutine CONT3D

Description

CONT3D plots contour levels of a function on a three-dimensional mesh. At the users option also the mesh may be plotted as well as a background consisting of parallel lines on a short distance.

Call

```
CALL CONT3D ( NUMBER, KMESH, KPROB, ISOL, CONTLN, NCNTLN, PLOTFM
              YFACT, ICHOICE, V )
```

Parameters

INTEGER NUMBER, KMESH(∗), KPROB(∗), ISOL(∗), NCNTLN, ICHOICE(∗)

DOUBLE PRECISION CONTLN(∗), PLOTFM, YFACT, V(3)

NUMBER, KMESH, KPROB, ISOL, PLOTFM, YFACT, CONTLN, NCNTLN See subroutine PLOTC1 (16.4.2).

ICHOICE Array of length 12 that must be filled by the user in the following way:

1 Highest entry number of ICHOICE that has been filled by the user. The positions 1 to ICHOICE(1) must have been given a value by the user. For all other positions default values are used. If IPLOTS(1)=0 or 1 only defaults are used.

2 Indication if the user defines the view position by filling array V (1) or not (0).

3 Indication if background lines must be plotted (1) or not (0).

4 Sequence number of background colour.

5 Indication if constituting lines must be plotted. Possible values:

- 0 Constituting lines are not plotted.
- 1 All constituting lines are plotted.
- 2 Only the visible parts of these lines are plotted.

6 Sequence number of colour of constituting lines.

7 Plot projection of three-dimensional mesh (1) or not (0).

8 Sequence number of colour of visible parts of mesh.

9 Sequence number of colour of invisible parts of mesh. If 0 these parts are not plotted.

10 Plot contour lines in projection plot (1) or not (0).

11 Colour of visible parts of contour lines.

12 Colour of invisible parts of contour lines. If 0 these parts are not plotted.

V If ICHOICE(2)=1 the user must fill the co-ordinates of the point where the observer is positioned in array V in the sequence x, y, z-co-ordinate.

Input

The parameters NUMBER, NCNTLN, PLOTFM and YFACT must have a value.
The arrays KMESH, KPROB, ISOL, CONTLN, ICHOICE and V must have been filled.

Output

Contour plot provided with extras at the users wish.

16.5 Vector plots (subroutine PLOTVC)

In this section we describe the vector plotting subroutines.
At this moment we have the following subroutines:

PLOTVC Original vector plotting subroutine

PLOTVCBF New vector plotting subroutine, extended with BUFFER and IBUFFR and some extra possibilities.

16.5.1 Subroutine PLOTVC

Description

A two-dimensional vector field with two components will be plotted. Each vector is represented by an arrow with direction and length in accordance with the value of the vector.
The vector can be derived from either one real, two real, or one complex standard SEPRAN vector.
Automatic scaling is possible.

Heading

```
subroutine plotvc ( idgfd1, idgfd2, ivctr1, ivctr2, kmesh, kprob,
                    plotfm, yfact, factor )
```

Parameters

INTEGER IDGFD1, IDGFD2, IVCTR1(5), IVCTR2(5), KMESH(∗), KPROB(∗)

DOUBLE PRECISION PLOTFM, YFACT, FACTOR

IDGFD1, IDGFD2 When both IDGFD1 and IDGFD2 are positive, the vector to be plotted is composed of the IDGFD1$^{\text{th}}$ and IDGFD2$^{\text{th}}$ degree of freedom of IVCTR1.
When both IDGFD1 and IDGFD2 are negative, the vector to be plotted is composed of the $-$IDGFD1$^{\text{th}}$ degree of freedom of the input vector IVCTR1 and the $-$IDGFD2$^{\text{th}}$ degree of freedom of IVCTR2.
If the input vector IVCTR1 corresponds to a complex array, the vector to be plotted is composed of the real and imaginary part of the $-$IDGFD1$^{\text{th}}$ degree of freedom of IVCTR1.
If in a nodal point not both degrees of freedom are present, the nodal point is skipped.

IVCTR1, IVCTR2 Arrays containing information of the vectors to be computed. Only when IDGFD1 and IDGFD2 are negative, and IVCTR1 corresponds to a real SEPRAN array, both arrays are used, otherwise only IVCTR1 is utilized. See IDGFD1 and IDGFD2.
IVCTR1 and IVCTR2 may be arrays of different type.

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

KPROB Standard SEPRAN array, containing information of the problem definition.

PLOTFM Length of plot in centimeters. Double precision variable, hence a double precision real must be submitted in this place.

YFACT Scale factor; all y-co-ordinates are multiplied by this factor before plotting the boundaries. Double precision real variable, hence a double precision real must be submitted.
YFACT $\neq$ 1 is used when the co-ordinates in x and y-direction are of different scales, and hence the picture becomes too small.

FACTOR Scale factor to be applied to the vector.
If FACTOR = 0, the scale factor is computed by PLOTVC, otherwise (FACTOR $>$ 0), the length of each vector is multiplied by FACTOR before plotting. For the length of the vector, the physical units are used, where the unit length is made equal to the geometrical unit length as indicated by the co-ordinates.

Input

The arrays KMESH, KPROB, IVCTR1 and when IDGFD1, IDGFD2¡ 0, IVCTR2 must be filled.
The parameters IDGFD1, IDGFD2, PLOTFM, YFACT and FACTOR must have a value.

Output

Plot of the vector field.

Remark

Suppose that the length and width of region to be plotted are of order 1000 units. Suppose furthermore that a typical length of the vector to be plotted is 3 units. Then a value of FACTOR of 30, introduces arrows to be plotted with a length of 0.09 times the length of the plot.

The computation of FACTOR

If on input FACTOR = 0, then PLOTVC computes the scaling factor in order to increase or decrease the length of the vector up to the same order as the distance between nodal points. FACTOR is the geometrical mean of two quantities:

(i) AMIN is minimum of the distance between point IPOINT on an element side and other points on the same element side, divided by the length of the vector to be plotted in IPOINT. All points inside the rectangle $(XMIN, XMAX) \times (YMIN, YMAX)$ are taken into account (see common block CPLOT (Section 16.2).

(ii) AMAX is maximum distance between a point on an element side and other points on the same element side, divided by the maximum length of the vector to be plotted. The points inside the rectangle $(XMIN, XMAX) \times (YMIN, YMAX)$ are taken into account.

Extra possibilities with help of common blocks

Of the extra possibilities which can be realized with the help of common blocks CPLOT and CGRENS (see Section 16.2), the restriction to a region outside which nothing is computed nor plotted, and the plotting of more vector fields in the same picture are the most important. See the description of the parameters JMAX and JTIMES in 16.2. If JTIMES $\geq$ 1 is used and if your installation has a plotter with more than one pen, a subroutine like CALCOMP'S NEWPEN can be called **after** each call of PLOTVC with JTIMES = 1 or 2. Consult your local System Manager.

16.5.2 Subroutine PLOTVCBF

Description

PLOTVCBF has exactly the same task as PLOTVC. The main difference is the parameters list and some extra possibilities available in this subroutine.

Heading

```
subroutine plotvcbf ( ibuffr, buffer, iplots, ivctr1,
                      ivctr2, kmesh, kprob, rplots )
```

Parameters

INTEGER IBUFFR(∗), IVCTR1(5), IVCTR2(5), KMESH(∗), KPROB(∗), IPLOTS(∗)

DOUBLE PRECISION BUFFER(∗), RPLOTS(∗)

IVCTR1, IVCTR2, KMESH, KPROB See subroutine PLOTVC

IBUFFR Standard integer buffer array as stored in blank common. The reason to put this parameter in the parameter list is that in this way blank common may be avoided.

BUFFER Standard double precision buffer array. I fact this is exactly the same array as IBUFFR.

IPLOTS Integer control array in which the user defines which choices he wants for this subroutine.
IPLOTS must be filled by the user in the following way:

Pos. 1 Defines the highest entry number of the array that has been filled by the user. This means that the user must fill all positions 1 to IPLOTS(1) in array IPLOTS. For all other positions default values are used. Hence if IPLOTS(1) = 0 or 1 only defaults are used.
Remark: if IPLOTS(i)=0, also for position i the default is taken.

Pos. 2 idgfd1, see PLOTVC.
Default value: 1

Pos. 3 idgfd2, see PLOTVC.
When both IDGFD1 and IDGFD2 are positive, the vector to be plotted is composed of the IDGFD1-th and IDGFD2-th degree of freedom of IVCTR1. When both IDGFD1 and IDGFD2 are negative, the vector to be plotted is composed of the -IDGFD1-th degree of freedom of the input vector IVCTR1 and the -IDGFD2-th degree of freedom of the input vector IVCTR2. If the input vector IVCTR1 corresponds to a complex array, the vector to be plotted is composed of the real and imaginary part of the IDGFD1-th degree of freedom of IVCTR1. If in a nodal point not both degrees of freedom are present, the nodal point is skipped.
Default value: 2

Pos. 4 inner, indication whether inner boundaries must be plotted (1) or not (0) (2D only)
Default value: 0

Pos. 5 Indicates if the computed factor must be stored in rplots(3) at output (1) or not (0)
Default value: 0

RPLOTS Double precision input array that must be filled as follows:

Pos. 1 plotfm Length of plot in centimeters.

Pos. 2 yfact Scale factor; all coordinates are multiplied by this factor before plotting.

Pos. 3 factor Scale factor to be applied to the vector. If FACTOR=0, the scale is computed, otherwise the length of each vector is multiplied by FACTOR before plotting.

Input

The arrays KMESH, KPROB, IVCTR1, IPLOTS, RPLOTS and when IDGFD1, IDGFD2¡ 0, IVCTR2 must be filled.
The arrays IBUFFR and BUFFER must have been filled.

Output

Plot of the vector field.
The arrays IBUFFR and BUFFER have been changed.

16.6 Function plots (subroutines PLOTFN and PLOTLN)

In this Section the subroutines that may be used for plotting of functions are treated. It concerns the following subroutines:

PLOTFN Make a plot of one-dimensional functions with an axis in one picture. These may be user defined functions or functions defined along curves.

PLOTLN intersects a 2D mesh with a straight line and computes the points in which the sides of the elements intersect the straight line. The input vector is interpolated in these intersection points and plotted.

Subroutine PLOTFN

Description

This subroutine plots one-dimensional functions with an axis in one picture. These function values are found along curves indicated by the user. At the users choice he can give the values to be plotted along the axis, or these values are plotted automatically.

Call

```
CALL PLOTFN ( ICHOICE, IPICT, NPICT, KMESH, KPROB, IVECTR, NUMBER, ICURVS,
              PLOTFM, YFACT, TEXTX, TEXTY, FUNCX, FUNCY )
```

Parameters

INTEGER ICHOICE, IPICT, NPICT, KMESH(∗), KPROB(∗), IVECTR(5), NUMBER, ICURVS(∗),

DOUBLE PRECISION PLOTFM, YFACT, FUNCX(∗), FUNCY(∗)

CHARACTER ∗ (∗) TEXTX, TEXTY

ICHOICE Choice parameter.
ICHOICE consists of three parts: ICHFUN, ISYMBOL and ICHPICT where the value of ICHOICE is equal to ICHFUN + 100 × ISYMBOL + 100000 × ICHPICT.

ICHFUN Choice parameter indicating how the function is given. Possibilities:

0 Standard situation. The NUMBER$^{\text{th}}$ degree of freedom with respect to array IVECTR in each nodal point of the curves indicated by array ICURVS is plotted. The x-co-ordinate of the picture is the arc length, starting from the initial point that is defined as x=0.
When IVECTR is a special array of type 119, the x and y co-ordinates of array IVECTR are used.

1 The x and y-co-ordinates of the function to be plotted are stored in the user arrays FUNCX and FUNCY. Array ICURVS is not used.

2 See ICHFUN = 0, however, the minimum and maximum x and y values XMIN, XMAX, YMIN and YMAX indicating the boundaries of the plot, are not computed from the functions to be plotted, but from the mesh boundaries. This possibility may only be used when IVECTR is a special array of type 119.

3 See ICHFUN = 2; the initial and end points of the curves to be plotted are connected by a straight line. This possibility is meant for the plotting of velocity profiles.

4 See ICHFUN = 3, the initial- and end-points of curves to be plotted are connected by a straight line.

ISYMBOL Choice parameter indicating with what symbols the functions are plotted. ISYMBOL may be different for each call of PLOTFN.
When ISYMBOL = 0, only straight lines are plotted.
When $1 \leq$ ISYMBOL ≤ 10 the points are connected to a line and marked with a symbol.
When $11 \leq$ ISYMBOL ≤ 20 the lines between points are neglected. Furthermore this situation is equal to ISYMBOL - 10.
Inquire at your local installation officer which symbols are available for
$1 \leq$ ISYMBOL ≤ 10

ICHPICT If ICHPICT > 0, then the right-axis of the picture is used as second y-axis with separate text and graduated scale.

IPICT Sequence number of the functions to be plotted in one picture. $1 \leq IPICT \leq NPICT$.

NPICT Number of functions to be plotted in one picture.

KMESH Standard SEPRAN array; output of the mesh generator.

KPROB Standard SEPRAN array, containing information of the problem definition.

IVECTR Array IVECTR is only used when ICHFUN = 0. In that case it must be the array containing information of the function to be plotted. IVECTR may be for example the solution vector ISOL or the output vector IOUTVC of subroutine DERIVA.

NUMBER The NUMBER$^{\text{th}}$ degree of freedom in each nodal point with respect to the array corresponding to $IVECTR$ is used for the plot.

ICURVS Input array to be defined and declared by the user.
ICURVS is only used when ICHFUN = 0.
ICURVS must be filled as follows:

ICURVS(1): Indication of how the curves are defined by the user. Possibilities:

ICURVS(1)=0: The function is plotted along one curve only. This curve number must be stored in ICURVS(2).
When ICURVS(2)<0 this curve is considered in reversed order.

ICURVS(1)=-1: The function is plotted along the curves ICRV1 to ICRV2 in that order, where the first point of ICRV1 is used as initial point.
ICRV1 must be stored in ICURVS(2).
ICRV2 must be stored in ICURVS(3).
The curves ICRV1 to ICRV2 must be subsequent curves. When ICRV2 < ICRV1, the sequence is given by ICRV1, ICRV1-1, . . . , ICRV2.

ICURVS(1)=-2: See ICURVS(1)=-1, however, the curves are used in reversed order. Hence the last point of ICRV1 is used as initial point.

ICURVS(1)=n >0: The function is plotted along the curves ICRV1, ICRV2, . . . , ICRVn in that order, where the first point of ICRV1 is used as initial point. The curve numbers ICRV1, ICRV2, . . . , ICRVn must be stored in ICURVS(2) , . . . , ICURVS(n+1).
When a negative curve number is given, the curve is used in reversed order. The curves must be subsequent curves.

PLOTFM Length of the plot in the x-direction.
When PLOTFM<0 the length of the plot in the x-direction in cm is equal to
- PLOTFM Δx, where Δx is the length of the curves.

YFACT Scale factor.
When YFACT>0 the height of the plot is equal to
|PLOTFM| $\times$ YFACT $\times \frac{\Delta y}{\Delta x}$ where Δy denotes the difference between the maximum and minimum function values to be plotted along the curves.
When YFACT<0 the height of the plot is equal to - YFACT cm.
When YFACT = 0, YFACT = 1 is assumed.

TEXTX,TEXTY Text to be plotted along the x-axis and. y-axis respectively.

FUNCX,FUNCY These parameters are only used when ICHFUN = 1.
FUNCX and FUNCY are variable length user arrays of type 117 and hence must be declared by the user.
Position 1 of these arrays must be filled by the user with the declared length.
FUNCX and FUNCY must be filled as follows:

1: Declared length of the arrays.

4: FUNCX: 1 and FUNCY: number of degrees of freedom per point (NUNKP).

5: Number of data in the arrays (following position 5), i.e.
FUNCX: number of nodal points (m=n).
FUNCY: number of nodal points times number of degrees of freedom per point. (m=n×NUNKP).

6-5+m: Actual data for the plot. Array FUNCX must contain the x-values sequentially, whereas FUNCY must contain the y-values in the sequence: u_1, v_1, w_1, u_2, v_2, w_2, . .
The points (FUNCX(i+5), FUNCY(5+(i-1)*NUNKP+NUMBER) (i=1,2, . . .,n) are connected to create a function plot.

Input

ICHOICE, IPICT, NPICT, NUMBER, PLOTFM, YFACT, TEXTX and TEXTY must have a value.
The arrays KMESH, KPROB, IVECTR and ICURVS must be filled.

Output

Plot of the picture with an axis when IPICT = NPICT.

Remarks

The subroutine computes the values along the axis automatically. However, when the user wants to use his own values, he must submit common block CGRENS as described in Section 16.2 in his main program.
The parameters KIESX, KIESY indicate whether the user gives the values to be plotted by the subroutine or not.
When KIESX=-1 the values to be plotted along the x-axis are computed and rounded, when KIESX≥0 the user must give the minimum and maximum values AXMIN, AXMAX.
KIESX=0 indicates that these values are plotted in floating point format, KIESX>0 in fixed format.
KIESY has the same meaning for the y-axis and AYMIN, AYMAX. The number of digits to be plotted when KIESX≥0 resp.
KIESY≥0 are given by ISIGX resp. ISIGY. See Section 16.2.
Default values (initialized by the SEPRAN starting subroutines) for KIESX and KIESY are -1.
When the user wants to plot extra information in the picture he may submit a subroutine PLMORE. This subroutine is called by PLOTFN and has no parameters. In PLMORE the extra plot statements must be programmed.

Extra option

As extra option it is possible to suppress the numbers and graduated scales. To that end the parameter JMARK in common block CPLOT (See 16.2) is used.
JMARK has the following meaning:
JMARK = 100 × JPLOTXS + 1000 × JPLOTYS + 10000 × JPLOTY2S
with

JPLOTXS Indication if numbers and graduated scales of x-axis must be plotted (0) or suppressed (1)

JPLOTYS Indication if numbers and graduated scales of y-axis must be plotted (0) or suppressed (1)

JPLOTY2S Indication if numbers and graduated scales of second (right) y-axis must be plotted (0) or suppressed (1)

Default value: JMARK < 100, hence JPLOTXS $= 0$, JPLOTYS $= 0$ and JPLOTY2S $= 0$.

Subroutine PLOTLN

Description

This subroutine intersects a 2D mesh with a straight line and computes the points in which the sides of the elements intersect the straight line. The input vector (for example the solution vector, or a derivative vector) is interpolated in these intersection points. The computed values are plotted by subroutine PLOTFN.

PLOTLN actually replaces the calls of INTP2D (Section 18.4) and PLOTFN.

Call

```
CALL PLOTLN ( ICHOICE, IVALUE, VALUES, NUMBER, KMESH, KPROB, ISOL, IPICT,
              NPICT, PLOTFM, HEIGHT, TEXTX, TEXTY )
```

Parameters

INTEGER ICHOICE, IVALUE, NUMBER, KMESH(∗), KPROB(∗), ISOL(5), IPICT, NPICT,

DOUBLE PRECISION VALUES, PLOTFM, HEIGHT, TEXTX, TEXTY

ICHOICE Choice parameter consisting of three parts: ICHINT, ISYMBOL and ICHPICT according to ICHOICE = ICHINT + 100 × ISYMBOL + 100000 × ICHPICT, with:

ICHINT Choice parameter indicating how the intersection line is defined. Possible values:

1 The user defines a line by defining a starting point and an angle with respect to the positive x-axis under which the line must be drawn in the positive x-direction. The values of the starting point respectively the angle must be stored into the array VALUES as follows:

VALUES(1) = xstart

VALUES(2) = ystart

VALUES(3) = angle of rotation with respect to positive x-axis in degrees.

ISYMBOL Choice parameter indicating with what symbols the functions are plotted. ISYMBOL may be different for each call of PLOTLN.
When ISYMBOL = 0, only straight lines are plotted.
When 1 ≤ ISYMBOL ≤ 10 the points are connected with a line and marked with a symbol.
When 11 ≤ ISYMBOL ≤ 20 the lines between points are neglected. Furthermore this situation is equal to ISYMBOL - 10.
Inquire at your local installation officer which symbols are available for 1 ≤ ISYMBOL ≤ 10.

ICHPICT If ICHPICT > 0, then the right-axis of the picture is used as second y-axis with separate text and graduated scale.

IVALUE Not yet used.

VALUES Double precision array in which the user must store real information concerning the intersection line. For the correct contents see ICHOICE.

NUMBER The $NUMBER^{th}$ unknown in each nodal point in the array referred to by ISOL will be selected. When this unknown does not appear in a node, this point is skipped.
When ISOL corresponds to a complex vector then NUMBER = 1 corresponds to the real part of the first unknown in each nodal point and NUMBER = 2 to the imaginary part. In the same way NUMBER = 3 corresponds to the real part of the second unknown etc. Hence when the real part of the i^{th} unknown is required, NUMBER should have the value $2i$-1, for the imaginary part NUMBER should be equal to $2i$.

KMESH Standard SEPRAN array; output of the mesh generator.

KPROB Standard SEPRAN array, containing information of the problem definition.

ISOL Array containing information of the vector to be interpolated. This may be for example the solution vector, but also a vector of related quantities like the derivatives.

IPICT Sequence number of the functions to be plotted in one picture. $1 \leq IPICT \leq NPICT$.

NPICT Number of functions to be plotted in one picture.

PLOTFM Length of the plot in the x-direction in cm.
When PLOTFM < 0 the length of the plot in the x-direction in cm is equal to
- PLOTFM $*$ (XMAX - XMIN).

HEIGHT Scale factor in y-direction.
When HEIGHT < 0 the height of the plot is equal to
|PLOTFM| $\times$ HEIGHT $\times$ (XMAX-XMIN) / (YMAX-YMIN)
When HEIGHT > 0 the height of the plot is equal to HEIGHT $\times$ PLOTFM cm.
When HEIGHT $= 0$, HEIGHT $= 1$ is assumed.

TEXTX,TEXTY Text to be plotted along the x-axis and. y-axis respectively.

Input

The arrays VALUES, KMESH, KPROB and ISOL must have been filled.
The parameters ICHOICE, IPICT, NPICT, PLOTFM, HEIGHT, TEXTX, TEXTY and NUMBER must have a value.

Output

Plot of the picture with an axis when IPICT = NPICT.

16.7 Three dimensional plots of a 2D function (subroutines PLOT3D and PLOT3C)

In this section we treat the subroutines that may be used to make three dimensional plots of functions defined in R^2. At this moment two such subroutines are available:

PLOT3D (16.7.1) Makes three dimensional plot with hidden lines.

PLOT3C (16.7.2) Makes a three-dimensional plot by filling (coloured) polygons. Hidden lines arise by plotting from far to close. This subroutine may only be used for pictures at displays.

16.7.1 Subroutine PLOT3D

Description

This subroutine makes a three-dimensional plot of a two-dimensional solution. Lines that are invisible for the observer are not plotted. (hidden lines are removed).

Heading

```
subroutine plot3d ( number, kmesh, kprob, ivectr, base, height, angle, ichoice )
```

Parameters

integer number, kmesh(∗), kprob(∗), ivectr(5), ichoice

double precision base, height, angle

number The NUMBERth unknown in each nodal point in array IVECTR will be selected. When this unknown does not appear in a nodal point, this point is skipped. When array IVECTR corresponds to a complex vector then NUMBER = 1 corresponds to the real part of the first unknown in each nodal point and NUMBER = 2 to the imaginary part. In the same way NUMBER = 3 corresponds to the real part of the second unknown etc. Hence when the real part of the ith unknown is required, NUMBER should have the value
2 × i - 1, for the imaginary part NUMBER should be equal to 2 × i.

kmesh Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

kprob Standard SEPRAN array, containing information of the problem definition.

ivectr Array containing information of the function to be plotted. This may be the solution vector, but also a vector of related quantities. Only a vector in a two-dimensional region is permitted.

base Length of base of the plot in centimeters. Double precision variable, hence a double precision real must be submitted in this place.

If BASE < 0 the absolute value of BASE is used. BASE < 0 indicates that HEIGHT(2) is used.

height Array of length 2 with the following contents.

1 Height of the plot in centimeters.
HEIGHT(1) should be at least as large as BASE. If HEIGHT(1) is smaller than BASE, HEIGHT(1) gets the value of BASE internally.

2 Is only used if BASE < 0.
HEIGHT(2) contains a reference value for the height of the plot, i.e. the maximal value is equal to HEIGHT(2) instead of the maximal value of the function.

angle This parameter gives the angle under which the observer sees the plot.
$0 \leq$ ANGLE ≤ 360

ichoice Choice parameter consisting of two parts LINOPT and NSTEP according to the relation ICHOICE = LINOPT + 10 $\times$ NSTEP with:

nstep Indication of how many grid lines are used for the 3D-plot. The number of lines in each direction is equal to $(NSTEP+1)\times\sqrt{NPOINT}$, with $NPOINT$ the number of points in the mesh. The number of grid lines may influence the quality of the picture, however, the computing time increases considerably for increasing values of NSTEP.

linopt Direction in which the surface lines are drawn. Possibilities:

1 parallel to y-axis

2 parallel to x-axis

3 both directions

5 pictures with the options 1,2 and 3 are made sequentially.

Input

The parameters NUMBER, BASE, HEIGHT, ANGLE and ICHOICE must have a value. The arrays KMESH, KPROB and IVECTR must be filled.

Output

Three-dimensional plot with hidden lines removed.

Remarks

When a 3-D plot is required for only a part of the region of definition, the parameters JMAX, XMIN, XMAX, YMIN and YMAX in common block /CPLOT/ as described in Section 16.2 may be utilised. When JMAX = 1 the parameters XMIN, XMAX, YMIN and YMAX define the plot area even when the function is defined in a smaller part of the two-dimensional space. In that case the function to be plotted is extended with the value 0 outside the definition region.

16.7.2 Subroutine PLOT3C

Description

PLOT3C makes a three-dimensional plot of a function defined on a two-dimensional region by filling (coloured) polygons. The subroutine starts to plot at the farthest faces and successively plots all faces closer to the observer. The closest surface is plotted last. In this way a hidden line plot with coloured faces arises automatically. The colour indicates the distance to the observer. This procedure can **only** be applied on a non-static display. If a black and white display is used, the final plot is also a standard hidden line plot. This way of plotting is much faster than by using PLOT3D (16.7.1). However, a plot with a pen plotter can not be made. In order to get a hard copy, one needs a hard copy unit which copies the prints of the screen.

Heading

```
subroutine plot3c ( kmesh, kprob, isol, iplots, rplots )
```

Parameters

integer kmesh(∗), kprob (∗), isol(∗), iplots(∗)

double precision rplots(∗)

kmesh Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

kprob Standard SEPRAN array, containing information of the problem definition.

isol Array containing information of the function to be plotted. This may be the solution vector, but also a vector of related quantities.

iplots Integer array of length 10+NELGRP, where NELGRP denotes the number of element groups.
IPLOTS must be filled by the user in the following way:

1 NUMBER; The $NUMBER^{th}$ unknown in each nodal point in array ISOL will be selected. When this unknown does not appear in a nodal point, this point is skipped.
When array ISOL corresponds to a complex vector then NUMBER = 1 corresponds to the real part of the first unknown in each nodal point and NUMBER = 2 to the imaginary part. In the same way NUMBER = 3 corresponds to the real part of the second unknown etc. Hence when the real part of the i^{th} unknown is required, NUMBER should have the value 2 × i - 1, for the imaginary part NUMBER should be equal to 2 × i.

2 Choice parameter for the projection method to be used. Possible values:

1 perspective projection.

2 parallel projection.

3 Sequence number of the farthest colour to be used.

4 Sequence number of the nearest colour to be used. The colour numbers that may be used depend on the installation of your plotting package and the type of display to be used. On a black and white terminal only colour 0 may be used, on for example a 256-colour display the colours to be used may vary from 0 to 256. The colour of a specific face is computed by linear interpolation between nearest and farthest colour.

5 Indication if the boundaries of the faces must be plotted (1) or not (0). These boundaries are plotted in the standard plotting colour of the display (usually white). In fact these boundaries clearly indicate the hidden line figure. On a black and white display the colour of the faces should be black. Combined with white boundaries, this results in a classical hidden-line picture.

6 ISKIP; Indication if element groups must be excluded (1) or not (0). If ISKIP = 1, positions 11 to 10+NELGRP of array IPLOTS should be filled.

7-9 Not yet used. Fill these positions with zeros in order to be certain that future versions give the same pictures as the present one.

11,...,10 +NELGRP These positions are only used if ISKIP = 1.
In that case IPLOTS(10+*ielgrp*) indicates if element group *ielgrp* must be skipped (0) or must be taken into account (1).

rplots Double precision array of length 10.
RPLOTS must be filled by the user in the following way:

1 PLOTFM; Length of plot in centimeters.

2 YFACT; Scale factor; all function values are multiplied by this factor before plotting.

3-5 Co-ordinates of the eye-point, i.e. the point where the observer is positioned. The co-ordinates must be stored in the sequence x, y, z.

6-10 Not yet used.

Input

The arrays KMESH, KPROB, ISOL, IPLOTS and RPLOTS must be filled.

Output

Coloured hidden-line plot on a display.

16.8 Particle tracing (subroutine PARTICLE)

In this section we treat the subroutines that are available for particle tracing. At this moment it concerns the following subroutines.

PARTICLE (16.8.1) Is the most general particle trace subroutine. It allows printing and plotting of the particle trajectories, but also output of the co-ordinates is possible. Besides that the user may influence the process by a function subroutine FNPARTIC (16.8.3).

PARTIC (16.8.2) Is only able to plot or print the particles. It is the predecessor of PARTICLE.

16.8.1 Subroutine PARTICLE

Description

Subroutine PARTICLE may be used to carry out a particle trace. Depending on the parameters the subroutine computes and prints or plots a number of particle traces in a velocity field or in a given flow field with porosity. This last possibility is typically meant for ground water flow.
Besides it is possible to get the co-ordinates and times of the particle trajectory as output in a SEPRAN array of type 118.
The user may have some influence on the trace process by the use of subroutine FNPARTIC (16.8.3). This subroutine may also be used to make output in user suitable way.

Heading

```
subroutine particle ( kmesh, kprob, isol, iinpar, rinpar, startp, iuser, user, ioutput )
```

Parameters

integer kmesh(∗), kprob(∗), isol(5), iinpar(7), iuser(∗), ioutput(5,∗)

double precision rinpar(5), user(∗), startp(ndim,∗)

kmesh Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

kprob Standard SEPRAN array, containing information of the problem definition.

isol Standard SEPRAN array of length 5 containing information of the velocity field or flow field. ISOL may be a solution array or an array of special structure containing derived quantities (possibly defined per element).

iinpar Integer array of length 10 to be filled by the user with information concerning the output.
IINPAR must be filled as follows:

1. Highest entry number of the array that has been filled by the user. The user must fill all positions 1 to IINPAR(1) in array IINPAR. For all other positions default values are used. Hence, if IINPAR(1)=0 or 1 only defaults are used.
2. NPART, i.e. number of particles to be traced.
   Default value: 1.
3. This parameter indicates if the trajectories must be printed or not.
   Possible values:
   -1 The trajectories are not printed.
   0 The trajectories are printed to the standard output file.

>0 The trajectories are printed to a file with reference number IINPAR(3). This file must have been opened before.

Default value: -1.

4. Indicates how the plots must be made.
   Possible values:
   0 No plots are made.
   1 The particle trajectories are plotted.
   2 Both the mesh and the trajectories are plotted (2D only).
   Default value: 1.
5. Indicates the type of parallel projection (3D only).
   Possible values:
   1 Projection on (x,z) plane from $y = \infty$.
   2 Projection on (y,z) plane from $x = \infty$.
   3 Projection on (x,z) plane from $y = -\infty$.
   4 Projection on (y,z) plane from $x = -\infty$.
   Default value: 1.
6. Indicates if the vector to be plotted is a standard velocity vector or that fluxes are given.
   Possible values:
   0 The vector is a standard velocity vector defined per nodal point or element. No porosity is given.
   1 The vector is a flow vector corresponding to a porous medium. In this case also the porosity is required and the arrays IUSER and USER must be used.
   2 See 0, however, now fluxes are defined through the edges instead of the velocity vector. This is for example the case if element type 650 is used.
   3 See 2, however, combined with porosity as described in 1.
   Default value: 0.
7. Indicates if markers must be plotted at the trajectories at fixed time-intervals.
   Possible values:
   0 No markers are plotted.
   1 Markers are plotted with a time step of tstep_mark, as stored in RINPAR(5).
   No other values are yet available.
   Default value:0.
8. Indicates if array IOUTPUT must be filled with information about the particle trajectories. Possible values:
   0 Array IOUTPUT is not used.
   1 Array IOUTPUT is filled by the subroutine.
   2 Array IOUTPUT is filled by the subroutine using the parameter tout_step as stored in RINPAR(7).
   Default value: 0
9. Indicates if for each position in the particle trace a user subroutine FNPARTIC (16.8.3) must be called (1) or not (0). With this subroutine the user may have some influence on the particle tracing.
   The description of this function subroutine can be found at the end of this section.
   Default value: 0
10. Indicates if tstep_print (RINPAR(6)) is used for the printing of the particles (1) or not (0).
    If tstep_print is not used the time step is detected by the program, which

implies that it is rather arbitrary and not constant.
Default value: 0

11. Defines type of time integration.
    Possible values:
    (a) Heun (explicit)
    (b) Runge Kutta (fourth order, explicit)
    (c) Trapezoid rule (implicit)
    Default value : 1
12. Indicates if EPS_TIME_STEP is given to define the accuracy for the determination of the time step (1) or not (0)
    Default value : 0

rinpar Double precision array of length 8 to be filled by the user with the following information:

1. PLOTFM Factor defining the length of the plot. When PLOTFM < 0 the length of the plot is compiled by -(xmax-xmin)∗PLOTFM and the width by -(ymax-ymin)∗PLOTFM∗YFACT where xmin, xmax, ymin and ymax are the minimum resp. maximum x and y values of the mesh.
   If PLOTF > 0 the length of the plot is equal to PLOTFM cm's.
   If |PLOTFM| $<$ 1cm the program uses the default length defined for the specific computer.
2. YFACT Multiplication factor for the y-co-ordinates. This factor influences the scaling in y-direction.
3. TBEGIN Starting time for tracing of particles.
4. TEND End time for tracing of particles.
5. TSTEP_MARK Indicates at which time intervals markers must be placed. This value is only used if IINPAR(7)>0.
6. TSTEP_PRINT Time interval for printing.
   Only used if iinpar(10) > 0
7. TSTEP_OUT Time interval for output.
   Only used if iinpar(8) > 1.
8. EPS_TIME_STEP, accuracy to define time step implicitly.
   Only used if iinpar(12) $= 1$, otherwise EPS_TIME_STEP $= 0.01$.

startp Two-dimensional double precision array of size $NDIM \times NPART$, where $NDIM$ is the dimension of space and $NPART$ is given by IINPAR(2). STARTP must be filled by the user with the co-ordinates of the starting points of the particles, in the following way:

x_i = startp (1,i)
y_i = startp (2,i)
z_i = startp (3,i)

iuser,user In these arrays the user must put the information with respect to the porosity per element group if IINPAR(6)=1 or 3. For each element group exactly one value is needed. IUSER and USER may be filled in the standard way using the subroutine FILCOF (6.2.1) or FIL100.

ioutput memory management number of array containing the particle trajectories at output.
The corresponding output array consists of an integer part and a real part.
The first position in the integer part contains npart, i.e. the number of particles.
The second position contains ndim (dimension of space).
This is followed by a two x npart integer array containing for each particle the number of positions for the particle trace (nsteps) and the starting address of the second dimension of the real array (starting from 1).

This part is followed by the real part The real array contains for each particle a (ndim+1) x nsteps real array, containing for each position t,x,y,z

Input

The arrays KMESH, KPROB, ISOL, IINPAR, RINPAR, STARTP and if necessary the arrays IUSER and USER must have been filled.

Output

Print and or plot of the particle trajectories.
Possibly array IOUTPUT has been filled.

16.8.2 Subroutine PARTIC

Description

Subroutine PARTIC has exactly the same task as PARTICLE (16.8.1). However, it does not contain array IOUTPUT, nor does it allow the use of function subroutine FNPARTIC (16.8.3).

Heading

```
subroutine partic ( kmesh, kprob, isol, iinpar, rinpar, startp, iuser, user )
```

Parameters

integer kmesh(∗), kprob(∗), isol(5), iinpar(7), iuser(∗)

double precision rinpar(5), user(∗), startp(ndim,∗)

kmesh, kprob, isol, rinpar, startp, iuser, user see PARTICLE (16.8.1).

iinpar Integer array of length 7 to be filled by the user with information concerning the output.
The position 1 to 7 have exactly the same meaning as for PARTICLE, positions 8 and 9 are not available.

Input

The arrays KMESH, KPROB, ISOL, IINPAR, RINPAR, STARTP and if necessary the arrays IUSER and USER must have been filled.

Output

Print and or plot of the particle trajectories.

16.8.3 User subroutine FNPARTIC

Description

FNPARTIC is a user written subroutine that is called each time a new position in a particle trajectory is computed, provided subroutine PARTICLE (16.8.1) has been called with IINPAR(9)=1.
This subroutine may be used to print the particle information in a user suitable way, or to store the information in a user common block.
Furthermore by means of this subroutine, the user may end the computation of the present particle trajectory.

Heading

```
subroutine fnpartic ( t, x, y, z, firstcall, continu )
```

Parameters

double precision t, x, y, z

logical firstcall, continu

t Time corresponding to the present position.

x,y,z Co-ordinates of the present position of the particle.

firstcall Input parameter, that may **not** be changed by the user. If true it concerns the first point in the trajectory, otherwise it is some other point.

continu This parameter indicates if the computation of the particle trajectory must be continued (`true`) or stopped (`false`). At input CONTINU has a value. The user may set this value to `false` if he wants to stop the computation of the present trace. He may never change the value from `false` to `true`.

Input

The parameters T, X, Y, Z, FIRSTCALL, CONTINU have a value.

Output

The user may have changed the value of CONTINU into `false`

Layout

Subroutine FNPARTIC must have the following shape:

```
      subroutine fnpartic ( t, x, y, z, firstcall, continu )
      implicit none
      double precision t, x, y, z
      logical firstcall, continu

c     Statements to produce user output
c     Statements to set continue to .false. in case the computation of
c     the trace of this particle must be stopped

      end
```

16.9 Plotting electric field lines (subroutine PFIELD)

Description

This subroutine makes a two-dimensional plot of the (electric-) field lines. The distance between the starting points of two consecutive field lines (Pi and Pi+1) is such that the flux-cut equals the (by the user supplied) value F:

$$F = \epsilon \int_{P_i}^{P_{i+1}} E_n ds$$

where ϵ is the dielectric constant and E denotes the normal component of the electric field.

Heading

```
subroutine pfield ( kmesh, kprob, user, iuser, isol, ichois, rchois,
                    rlengs, nrofln )
```

Parameters

integer kmesh(∗), kprob(∗), iuser(∗), isol(∗), ichois(3), nrofln

double precision user(∗), rchois(5), rlengs(∗)

kmesh Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

kprob Standard SEPRAN array, containing information of the problem definition.

iuser,user In these arrays information concerning the coefficients etc. must be stored. Consult the manual 'Standard Problems' for the actual filling of these arrays. The first coefficient is taken to be the dielectric constant. These arrays are only used when ICHOIS(1) = 1,2 or 3.

isol Standard SEPRAN array with the structure of a solution array containing information of the electric field.

ichois Integer array of length 3 that must be filled by the user in the following way:

1 Indicator to plot field lines. Possible values:
  1 Make general plot of field lines.
  2 Make plot of field lines starting on some part of the boundary.
  3 Make plot of field lines ending on some part of the boundary.
  4 Make plot of field lines through point X.
  5 Make plot of field lines starting in point X.
  6 Make plot of field lines ending in point X.

2 First user point of boundary part. Only used if ICHOIS(1)=2 or 3.

3 Last user point of boundary part. Only used if ICHOIS(1)=2 or 3.

rchois Double precision array of length 5 that must be filled by the user in the following way:

1 PLOTFM, length of the plot in centimeters.

2 YFACT, scale factor; all y-co-ordinates are multiplied by this factor before plotting the boundaries.

3 F, flux-cut between two points on consecutive field lines. Only used if ICHOIS(1) $\leq$ 3.

4 X1, x-co-ordinate of starting point. Only used if ICHOIS(1) > 3.

5 Y1, y-co-ordinate of starting point. Only used if ICHOIS(1) > 3.

rlengs Double precision array of length NROFLN, containing the lengths of the field lines.

nrofln At input: the length of RLENGS.
At output: the number of positions filled in RLENGS.

Since NROFLN is both input and output variable, NROFLN must be a variable and **not** a constant!

Input

The arrays KMESH, KPROB, USER, IUSER, ISOL, ICHOIS and RCHOIS must be filled.
NROFLN must have a value ≥ 0.

Output

Plot of field lines. The first NROFLN positions of RLENGS have been filled. The lengths of the field lines have been written to the standard output file.
NROFLN has got a new value.

16.10 Plot two-dimensional distorted mesh (subroutine PLOTDT)

Description

Plot a two-dimensional distorted mesh owing to the displacements.

Heading

```
subroutine plotdt ( idgfd1, idgfd2, kmesh, kprob, isol, plotfm, yfact, factor )
```

Parameters

integer idgfd1, idgfd2, isol(5), kmesh(∗), kprob(∗)

double precision plotfm, yfact, factor

idgfd1, idgfd2 The displacement $\mathbf{u} = (u\ ,\ v)^T$ is given by the IDGFD1$^{\text{th}}$ and IDGFD2$^{\text{th}}$ degree of freedom in the nodal points.

isol Array containing information concerning the solution vector.

kmesh Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

kprob Standard SEPRAN array, containing information of the problem definition.

plotfm Length of plot in centimeters. Double precision variable, hence a double precision real must be submitted in this place.

yfact Scale factor; all y-co-ordinates are multiplied by this factor before plotting the boundaries. Double precision real variable, hence a double precision real must be submitted.
YFACT $\neq$ 1 is used when the co-ordinates in x and y-direction are of different scales, and hence the picture becomes too small.

factor The co-ordinates of the distorted mesh are given by:
$x_{new} = x + \text{FACTOR} \times u$
$y_{new} = y + \text{FACTOR} \times v$

Input

The arrays KMESH, KPROB and ISOL must have been filled.
IDGFD1, IDGFD2, PLOTFM, YFACT and FACTOR must have a value.

Output

Plot of the distorted mesh.

16.11 Plot curves (subroutine PLOTCU)

Description

Plot the curves in a two-dimensional mesh.

Heading

```
subroutine plotcu ( ibuffr, buffer, kmesh, iplots, rplots, textx, texty )
```

Parameters

integer ibuffr(∗), kmesh(∗), iplots(∗)

double precision buffer(∗), rplots(∗)

character ∗(∗) textx, texty

IBUFFR Standard integer buffer array as stored in blank common. The reason to put this parameter in the parameter list is that in this way blank common may be avoided.

BUFFER Standard double precision buffer array. In fact this is exactly the same array as IBUFFR.

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

IPLOTS Integer array of length 10 to be filled by the user with the following information:

1 Indication if user points (1) or curves must be plotted (2)

2 IDOUBLE, indication whether the boundaries are plotted with double lines (IDOUBLE = 1) or not (IDOUBLE = 0).

3 IINNER, Information of how the curves must be plotted.
Possible values

1 only the outer boundaries are plotted

2 only the inner boundaries are plotted

3 all boundaries are plotted

4 NUMBER, Indication if the curves must be provided with a curve number (1) or not (0).

5 ISYMBL, Indicates if the nodes along the curves must be marked with a cross (1) or not (=0)

6-10 Not yet used (Fill zeros for future developments).

RPLOTS Array to be filled by the user with the following quantities:

1. 1] PLOTFM (length of the plot in centimeters)
2. 2] YFACT (multiplication factor for the y-co-ordinates)

TEXTX Text to be plotted along x-axis

TEXTY Text to be plotted along y-axis

Input

The arrays KMESH, RPLOTS and IPLOTS must have been filled.
TEXTX and TEXTY must have a value.
The arrays IBUFFR and BUFFER must have been filled.

Output

Plot of the curves.

17 Free-surface and moving boundary problems

17.1 Introduction

In this section we shall describe some subroutines that are especially meant for free surface and moving boundary problems. Such problems can be handled by program SEPFREE as described in the Users Manual. However, if the user wants to make his own program, since for example SEPFREE does not allow sufficient possibilities, then the subroutines treated in this chapter may of great help.
An important aspect of free and moving boundary problems is that in each step of the process it may be necessary to adapt the boundary and consequently the mesh. Subroutines dealing with these aspects have already been treated in Chapter 3. In order to use these subroutines it is common practice, although not necessary to create the mesh in the main subprogram and not separately by program SEPMESH. This means that the starting subroutine SEPSTM (4.2.4) (Section 4.2) must be used rather than SEPSTR (4.2.1) or one of its counterparts. The alternative is of course explicitly creating the mesh by subroutine MESH.
Once the boundary is changed, the mesh must be adapted. This may be done by changing the co-ordinates only, thus keeping the old topology. This is far most the cheapest solution, but it may be possible that after some time the quality of the mesh decreases considerably. If that is the case, remeshing is necessary and the topology is recomputed. Of course a new topology implies that also the problem description and the matrix structure must be recomputed. Subroutines to perform this task have already been described in Chapter 4.
If the topology is changed and the user needs the solution at the preceding mesh, it may be necessary to interpolate the solution from the old mesh to the new one. This may be done by subroutine INTMSH treated in Section 17.3.
For the global iteration process some help subroutines have been developed. At this moment such a subroutine is STATFREE, which is meant for stationary free boundary problems.

Hence with respect to free and moving boundary problems, the following subroutines are very important:

MESH Create a new mesh or adapt an existing mesh, see Section 3.2.

ADAPBOUN Adapt the boundary of a mesh, see Section 3.9.

ADAPMESH Adapt a mesh, see Section 3.10.

SEPSTM (4.2.4) General starting subroutine, creating an initial mesh, see Section 4.2.

STATFREE Help subroutine for the global iteration process in case of a stationary free boundary problem. See Section 17.2.

INTMSH Interpolate the solution from one mesh to another one. See Section 17.3.

MSHCOPY, PROCOPY Copy those parts of KMESH and KPROB that are necessary for the interpolation by INTMSH. See Section 17.3.

INSTFREE Help subroutine for the adaptation of the mesh in case of a instationary free boundary problem. See Section 17.4.

TIMEFREE Help subroutine for the time stepping process in case of a instationary free boundary problem. See Section 17.5.

17.2 Subroutine STATFREE, a tool for stationary free boundary problems

Description

Subroutine to be used for stationary free surfaces It checks for convergence and if necessary adapts the mesh. This is a very special subroutine in the SEPRAN context, since it assumes that the user has made a loop himself.
STATFREE should be called in this loop It returns with a value for ready indicating if the loop may be finished or not. The following tasks are performed:

- If niter > 1, the difference between the old and the new solution is computed.
  If this difference satisfies the convergence criterion the subroutine returns with ready set equal to true.
  At the users option the final mesh is written.
- If the subroutine has not been finished then the mesh is adapted using subroutine ADAPMESH (3.10). See subroutine ADAPMESH (3.10) for a description.

In contrast to ADAPMESH (3.10) input is only possible through the input file and not through the parameter list.

Call

```
CALL STATFREE ( NITER, READY, KMESH, KPROB, INTMAT, ISOL, IREAD )
```

INTEGER NITER, KMESH(∗), KPROB(∗), INTMAT(5,∗), ISOL(5,∗), IREAD

LOGICAL READY

NITER Iteration sequence number.
In the first call niter must have the value 1. It is raised each next iteration and reset to 1 if convergence has been achieved.
So NITER must be a variable and may never be a constant. Moreover, the user must give NITER the value 1 at the start of the process and this value is changed by the subroutine.

READY This parameter indicates if convergence has been reached or not. This is an output parameter hence it must be a variable and may never be a constant. As long as no convergence has been reached, the subroutine returns with READY equal to $false$. If the solution has been converged READY is set equal to $true$.

KMESH Standard SEPRAN array corresponding to the mesh.
At input kmesh corresponds to the mesh to be adapted, at output, kmesh corresponds to the new mesh

KPROB Standard SEPRAN array corresponding to the problem definition.
At input kprob must correspond to kmesh at input.
If the topology of the mesh is changed, kprob has been adapted at output.

INTMAT Standard SEPRAN array containing information of the matrix structure.
At input intmat must correspond to kmesh and kprob at input.
If the topology of the mesh is changed, intmat has been adapted at output.

ISOL Standard SEPRAN array containing information of the solution array at the present iteration. At output the solution array corresponding to ISOL may have been changed depending on the input.

IREAD Defines how the input for this subroutine must be read.
Possible values:

0 The input is read from the standard input file

> 0 The input as read from the standard input file by subroutine SEPSTM (4.2.4) is used. The value of iread defines the sequence number

Input

The arrays KMESH, KPROB, INTMAT and ISOL must have been filled.
The parameters NITER and IREAD must have a value.

The subroutine requires input from the standard input file. This is the input defined by the input block "STATIONARY_FREE_BOUNDARY". See the Users Manual Section 3.4.5.

Output

The value of NITER has been changed.
READY has got a value.
The mesh corresponding to KMESH may have been changed. Also the contents of the array KMESH may have been changed.
The arrays ISOL, INTMAT and KPROB may have been changed as well as the arrays corresponding to them.

How to use subroutine STATFREE

Subroutine STATFREE is a special subroutine in the sense that it is essentially part of a user written loop. So the user has to provide the loop and decide himself how to leave the loop.
The most common way to use STATFREE in a program is the following one:

```
      program freesurface

c     --- start of program including declarations

         .
         .
         .

c     --- start of loop:

      ready = .false.
      niter = 1

100   if ( .not. ready ) then

c     --- statements to compute the solution during one iteration

              .
              .
              .

         iread = ...
         call statfree ( niter, ready, kmesh, kprob, intmat, isol,
     +                   iread )
         go to 100

      end if
         .
         .
         .
         .
      end
```

17.3 Interpolation from one mesh into another one

The interpolation from one mesh to another is carried out by subroutine INTPOLMS or INTMSH. In order to do so, it is necessary to have the arrays KMESH and KPROB for both the old and the new mesh. Usually the mesh is adapted by a subroutine like MESH or ADAPMESH (3.10). As a consequence the old mesh is overwritten. So before creating the new mesh it is necessary to copy the old mesh and old problem description into help arrays. Since INTPOLMS and INTMSH use only a few parts of KMESH and KPROB it is only necessary to copy those parts. The subroutines MSHCOPY and PROCOPY may be used to perform this task.
In this section the following subroutines are treated:

INTMSH (17.3.1) Interpolate the solution from one mesh to another.

INTPOLMS (17.3.2) Extension of subroutine INTMSH.

INTCOOR (17.3.3) This subroutine interpolates the solution defined on a mesh onto a series of points of which the user defines the co-ordinates.

PRINGRID (17.3.4) This subroutine interpolates the solution defined on a mesh onto a user defined rectangular grid and prints the interpolated solution.

MSHCOPY (17.3.5) Copy relevant parts of array KMESH with respect to INTMSH.

PROCOPY (17.3.6) Copy relevant parts of array KPROB with respect to INTMSH.

17.3.1 Subroutine INTMSH

Description

This subroutine interpolates the solution defined on one mesh into a new solution array defined on another mesh.

Points within the preceding mesh or close to the old mesh get a value, the other points are made equal to zero.

Heading

```
subroutine intmsh ( ichoice, kmesh1, kmesh2, kprob1, kprob2, isol1, isol2 )
```

Parameters

INTEGER ICHOICE, KMESH1(∗), KMESH2(∗), KPROB1(∗), KPROB2(∗), ISOL1(5), ISOL2(5)

ICHOICE This parameter indicates which part of the solution is interpolated.
Possible values:

- 0,1 All points at the new mesh are interpolated from the old mesh.
  If points of the new mesh are outside the old one, an extrapolation is used.
- 2 Only those of the new mesh that are inside the old mesh or very close to old boundary are interpolated. All other points are kept at there old value or if array ISOL2 is new, are set to 0.

KMESH1 Standard SEPRAN array corresponding to the first mesh; output of the mesh generator.

KMESH2 Standard SEPRAN array corresponding to the second mesh; output of the mesh generator.

KPROB1 Standard SEPRAN array corresponding to the first problem definition.

KPROB2 Standard SEPRAN array corresponding to the second problem definition.

ISOL1 Array containing information of the solution vector to be interpolated. ISOL1 must correspond to the arrays KMESH1 and KPROB1.

ISOL2 Array ISOL2 contains information of the interpolated solution vector. ISOL2 corresponds to the arrays KMESH2 and KPROB2.

Input

The arrays KMESH1, KMESH2, KPROB1, KPROB2 and ISOL1 must have been filled. The parameters ICHOICE must have the value one.

Output

Array ISOL2 has been filled.

17.3.2 Subroutine INTPOLMS

Description

This subroutine interpolates the solution defined on one mesh into a new solution array defined on another mesh. Information concerning the formal mapping may be stored in an array denoted by MAP. In this way reuse of the subroutine for other solution vectors may imply a large gain in computing time.

In fact INTPOLMS is identical to INTMSH, except that it allows the reuse of interpolation information. So for a single interpolation both subroutines are completely identical.

Heading

```
subroutine intpolms ( iinmap, kmesh1, kmesh2, kprob1, kprob2,
                      isol1, isol2, map )
```

Parameters

INTEGER IINMAP(∗), KMESH1(∗), KMESH2(∗), KPROB1(∗), KPROB2(∗), ISOL1(5), ISOL2(5), MAP(5)

IINMAP Input array defined by the user.
IINMAP gives some extra information about the interpolation.
Array IINMAP must be filled as follows:

1. Last entry number that has been filled by the user. If the value is 0 or 1 only defaults are used.
2. infmap, indicates if and how array MAP is used.
   Possible values:
   - 0 Array MAP is not used. In this case subroutines INTMSH and INTPOLMS are identical.
   - 1 Information about the formal mapping is stored in array MAP. It may be reused in a next call of INTPOLMS, thus reducing the computation time considerably.
   - 2 Array MAP has been filled before for these two meshes and the information is reused for the mapping.

   Default value : 0
3. minmax, indicates if the interpolation is free (0) or that the interpolated values are restricted to the range of the original array. In that case we get: uold_iun(min) ≤ unew_iun ≤ uold_iun(max) for each physical unknown iun.
   Default value : 0

KMESH1, KMESH2, ISOL1, ISOL2, KPROB1, KPROB2 See subroutine INTMSH (17.3.1)

MAP Integer array of length 5 that contains information about the mapping. For a description of array MAP, see Section 24.9.
If infmap = 2 MAP will be filled.
If infmap = 3 MAP must have been filled before by a call to INTPOLMS with infmap = 2.

Input

The arrays IINMAP, KMESH1, KMESH2, KPROB1, KPROB2 and ISOL1 must have been filled.
Depending on the value of IINMAP(2) array MAP must have been filled.

Output

Array ISOL2 has been filled.
Depending on the value of IINMAP(2) array MAP has been filled.

17.3.3 Subroutine INTCOOR

Description

This subroutine interpolates the solution defined on a mesh into a new solution array defined on a set of points defined by the user. The user must give the co-ordinates of these points explicitly.
Information concerning the formal mapping may be stored in an array denoted by MAP. In this way reuse of the subroutine for other solution vectors may imply a large gain in computing time.

Heading

```
subroutine intcoor ( kmesh, kprob, isol, solint, coor, nunks,
                     ncoor, ndim, iinmap, map )
```

Parameters

INTEGER IINMAP(∗), KMESH(∗), KPROB(∗), ISOL(5), MAP(5), NUNKS, NCOOR, NDIM

DOUBLE PRECISION COOR(NDIM,NCOOR), SOLINT(NUNKS,NCOOR)

KMESH Standard SEPRAN array corresponding to the mesh.

KPROB Standard SEPRAN array corresponding to the problem description.

ISOL Standard SEPRAN array containing information of the solution.

SOLINT Output array in which the interpolated solution in the NCOOR points is stored.
SOLINT(i,j) contains the i^{th} degree of freedom in the j^{th} interpolation point.

COOR Input array in which the user must put the coordinates of the points in which the solution must be interpolated.
COOR(i,j) refers to the i^{th} co-ordinate in the j^{th} point.

NUNKS Number of degrees of freedom per point that must be interpolated. If this number is more than the maximum number of degrees of freedom per point, not more than this maximum number of values is filled.
Defines the size of array SOLINT.

NCOOR Number of points of which co-ordinates are given.
Defines the size of the arrays COOR and SOLINT.

NDIM Dimension of the space.
Is used as parameter to dimension array COOR.
At this moment only NDIM = 2, 3 has been implemented.

IINMAP Input array defined by the user.
IINMAP gives some extra information about the interpolation.
Array IINMAP must be filled as follows:

1 Last entry number that has been filled by the user. If the value is 0 or 1 only defaults are used.

2 infmap, indicates if and how array MAP is used.
Possible values:

0 Array MAP is not used.

1 Information about the formal mapping is stored in array MAP. It may be reused in a next call of INTCOOR, thus reducing the computation time considerably.

2 Array MAP has been filled before for these two meshes and the information is reused for the mapping.

Default value : 0

3 minmax, indicates if the interpolation is free (0) or that the interpolated values are restricted to the range of the original array. In that case we get: uold_iun(min) $\leq$ unew_iun $\leq$ uold_iun(max) for each physical unknown iun.
Default value : 0

MAP Integer array of length 5 that contains information about the mapping. For a description of array MAP, see Section 24.9.
If infmap = 1 MAP will be filled.
If infmap = 2 MAP must have been filled before by a call to INTCOOR with infmap = 1.

Input

The arrays IINMAP, KMESH, KPROB, ISOL and COOR must have been filled.
Depending on the value of IINMAP(2) array MAP must have been filled.
The parameters NCOOR, NDIM and NUNKS must have a value.

Output

Array SOLINT has been filled.
Depending on the value of IINMAP(2) array MAP has been filled.

17.3.4 Subroutine PRINGRID

Description

Interpolate the solution stored in array isol to a rectangular grid with bounds (xmin,xmax,ymin,ymax,zmin,zmax) and step size (dx,dy,dz).
Print the interpolated values

Heading

```
subroutine pringrid ( ibuffr, buffer, kmesh, kprob, isol, rinput,
                      text )
```

Parameters

INTEGER IBUFFR(∗), KMESH(∗), KPROB(∗), ISOL(5)

DOUBLE PRECISION BUFFER(∗), RINPUT(3)

CHARACTER ∗(∗) TEXT

IBUFFR Standard integer buffer array as available in blank common.

BUFFER Standard double precision buffer array as available in blank common.

KMESH Standard SEPRAN array corresponding to the mesh.

KPROB Standard SEPRAN array corresponding to the problem description.

ISOL Standard SEPRAN array containing information of the solution.

RINPUT Real input array containing the step sizes in the following way:

1. step size dx
2. step size dy
3. step size dz

text String variable containing the text for the heading

Input

The arrays IBUFFR, BUFFER, KMESH, KPROB and ISOL must have been filled.
The user must have filled array RINPUT.
The parameters TEXT must have a value.

Output

Print of the solution mapped to a rectangular grid.

17.3.5 Subroutine MSHCOPY

Description

Copy relevant parts of array KMESH into a new array. It concerns those parts of the mesh that are used by subroutine INTMSH (17.3.1).
MSHCOPY must be called before the new mesh is created.

Heading

```
subroutine mshcopy ( kmesh1, kmesh2 )
```

Parameters

INTEGER KMESH1(∗), KMESH2(∗)

KMESH1 Standard SEPRAN array corresponding to the mesh to be copied: output of the mesh generator.

KMESH2 Copy of array KMESH1. Those parts that are necessary for subroutine INTMSH are copied, including the parts stored in the buffer array IBUFFR.

Input

Array KMESH1 must have been filled.

Output

Array KMESH2 has been filled, as well as the parts in the buffer array IBUFFR, corresponding to it.

17.3.6 Subroutine PROCOPY

Description

Copy relevant parts of array KPROB into a new array. It concerns those parts of the problem definition that are used by subroutine INTMSH (17.3.1).
PROCOPY must be called before the new mesh is created.

Heading

```
subroutine procopy ( kprob1, kprob2 )
```

Parameters

INTEGER KPROB1(∗), KPROB2(∗)

KPROB1 Standard SEPRAN array containing information of the problem description. KPROB1 must correspond to the mesh to be copied.

KPROB2 Copy of array KPROB1. Those parts that are necessary for subroutine INTMSH are copied, including the parts stored in the buffer array IBUFFR.

Input

Array KPROB1 must have been filled.

Output

Array KPROB2 has been filled, as well as the parts in the buffer array IBUFFR, corresponding to it.

17.4 Subroutine INSTFREE, a mesh adapting tool for time-dependent free boundary problems

Description

Subroutine to be used for instationary free surfaces.
It adapts the free boundary This is a very special subroutine in the SEPRAN context, since it assumes that the user has made a loop himself.
INSTFREE should be called in this loop. Furthermore, INSTFREE must always be used in combination with subroutine TIMEFREE or a user written equivalent. That subroutine has to take care of the time-stepping algorithm.
In contrast to adapmesh input is only possible through the input file and not through the parameter list. The following tasks are performed:

- The boundary and the mesh are adapted.
- At the users request the mesh velocity is computed.

Call

```
CALL INSTFREE ( KMESH, KPROB, INTMAT, ISOL, IREAD )
```

INTEGER KMESH(∗), KPROB(∗), INTMAT(5,∗), ISOL(5,∗), IREAD

KMESH Standard SEPRAN array corresponding to the mesh.
At input kmesh corresponds to the mesh to be adapted, at output, kmesh corresponds to the new mesh.

KPROB Standard SEPRAN array corresponding to the problem definition.
At input kprob must correspond to kmesh at input.
If the topology of the mesh is changed, kprob has been adapted at output.

INTMAT Standard SEPRAN array containing information of the matrix structure.
At input intmat must correspond to kmesh and kprob at input.
If the topology of the mesh is changed, intmat has been adapted at output.

ISOL Standard SEPRAN array containing information of the solution array at the present iteration. At output the solution array corresponding to ISOL may have been changed depending on the input.

IREAD Defines how the input for this subroutine must be read.
Possible values:

- 0 The input is read from the standard input file
- > 0 The input as read from the standard input file by subroutine SEPSTM (4.2.4) is used. The value of iread defines the sequence number

Input

The arrays KMESH, KPROB, INTMAT and ISOL must have been filled.
Parameter IREAD must have a value.

The subroutine requires input from the standard input file. This is the input defined by the input block "INSTATIONARY_FREE_BOUNDARY". See the Users Manual Section 3.4.6.

Output

The mesh corresponding to KMESH may have been changed. Also the contents of the array KMESH

may have been changed.
The arrays ISOL, INTMAT and KPROB may have been changed as well as the arrays corresponding to them.
If required the mesh velocity has been stored in one of the arrays corresponding to ISOL.

17.5 Performing a time step in an instationary free boundary problem (subroutine TIMEFREE)

Description

Subroutine to be used for instationary free surfaces.
It performs one time-step.
This is a very special subroutine in the SEPRAN context, since it assumes that the user has made a loop himself. TIMEFREE should be called in this loop. It returns with a value for ready indicating if the loop may be finished or not. The following tasks are performed:

- In first call: set time integration constants
- Perform one time step
- In last time step: set ready and reset the local parameter first

In contrast to ADAPMESH (3.10) input is only possible through the input file and not through the parameter list.

Call

```
CALL TIMEFREE ( READY, KMESH, KPROB, INTMAT, ISOL, IINTIMF, NUMVEC )
```

INTEGER KMESH(∗), KPROB(∗), INTMAT(5,∗), ISOL(5,∗), IINTIMF(∗), NUMVEC

LOGICAL READY

READY This parameter indicates if the end time has been reached. This is an output parameter hence it must be a variable and may never be a constant. As long as $t < tend$, the subroutine returns with READY equal to $false$. If $t \geq tend$ READY is set equal to $true$.

KMESH Standard SEPRAN array corresponding to the mesh.
At input kmesh corresponds to the mesh to be adapted, at output, kmesh corresponds to the new mesh

KPROB Standard SEPRAN array corresponding to the problem definition.
At input kprob must correspond to kmesh at input.
If the topology of the mesh is changed, kprob has been adapted at output.

INTMAT Standard SEPRAN array containing information of the matrix structure.
At input intmat must correspond to kmesh and kprob at input.
If the topology of the mesh is changed, intmat has been adapted at output.

ISOL Standard SEPRAN array containing information of the solution array at the present iteration. At output the solution array corresponding to ISOL may have been changed depending on the input.

IINTIMF Integer input array with the following contents:

1. IREAD Defines how the input for this subroutine must be read.
   Possible values:
   - 0 The input is read from the standard input file
   - > 0 The input as read from the standard input file by subroutine SEPSTM (4.2.4) is used. The value of iread defines the sequence number
2. IVECTR, sequence number of solution vector with respect to array ISOL.

NUMVEC Largest sequence number of vectors that have been filled in array ISOL.

Input

The arrays KMESH, KPROB, INTMAT, IINTIMF and ISOL must have been filled.
Parameter NUMVEC must have a value.

The subroutine requires input from the standard input file. This is the input defined by the input block "TIME_INTEGRATION". See the Users Manual Section 3.4.6.

Output

In the first call common block CTIMEN has been initialized.
The value of the actual time t has been changed in common block CTIMEN.
READY has got a value.
Array ISOL may have been changed as well as the arrays corresponding to them.

How to use the combination of the subroutines TIMEFREE and INSTFREE

The subroutines TIMEFREE and INSTFREE are special subroutines, in the sense that are supposed to be used in combination and, moreover, are part of a user written loop.
So the user has to provide the loop and decide himself how to leave the loop.
The most common way to use these subroutines in a program is the following one:

```
      program freesurface

c     --- start of program including declarations

         .
         .
         .

c     --- start of loop:

      ready = .false.

100   if ( .not. ready ) then

c     --- statements to compute the solution during one iteration

             .
             .
             .

         iintimf(1) = ...
         iintimf(2) = ...
         call timefree ( ready, kmesh, kprob, intmat, isol, iintimf,
     +                   numvec )
             .
             .
             .
         iread = ...
         call instfree ( kmesh, kprob, intmat, isol, iread )
             .
             .
             .
         go to 100

      end if
         .
         .
         .
         .
      end
```

18 Mapping subroutines

18.1 Introduction

In this chapter we describe all types of subroutines to map SEPRAN vectors into other types of SEPRAN vectors, or into user defined vectors. This mapping may also imply mapping from a 3d region onto a 2d region or from 2d onto 1d.

The following mapping subroutines are available:

COPYUS(18.2.1) This subroutines maps a SEPRAN vector into a user vector or a user vector into a SEPRAN vector.

COPYUSBF(18.2.2) is the equivalent of COPYUS not using blank common.

PUTUSER(18.2.3) Puts values from a user array into a solution array.

PUTUSBF(18.2.4) is the equivalent of PUTUSER not using blank common.

COMPCR(18.3.1) Computes values of an array along user specified curves and maps them in a user array. Also the corresponding distances along the curves are stored.

COMPCRBF(18.3.2) is the equivalent of COMPCR not using blank common.

COMPSR(18.3.3) Computes values of an array along user specified surfaces and maps them in a user array. Also the corresponding coordinates along the surfaces and the corresponding node numbers are stored.

COMPSRBF(18.3.4) is the equivalent of COMPSR not using blank common.

INTP2D(18.4) Intersects a 2d mesh with a straight line. The intersection points of the mesh with the line are stored and the corresponding function values are found by interpolation in the solution array.

INTP3D(18.5) Has the same task as INTP2D, however, now it concerns the intersection of a plane with a three-dimensional mesh. In the plane a new mesh consisting of linear triangles is constructed.

TRANPL(18.6) is used to transform the co-ordinates and the solution from polar co-ordinates to Cartesian co-ordinates.

MAPCOM(18.7) Maps a SEPRAN matrix or its integer description, from one type of storage into another one. For example the mapping is from profile storage to compact storage.

18.2 Mapping of a SEPRAN vector into a user vector or vice versa.

In this section we treat the subroutines that are available to map the contents of SEPRAN arrays into user arrays and vice versa.
The following six subroutines are available

COPYUS With this subroutine, the user may copy the contents of a SEPRAN solution array or array of special structure, into a user array of vice-versa. Also the user may copy the contents of the solution array in a user point into a user array.

COPYUSBF Exactly the same subroutine as COPYUS, however, in this case blank common is not used.

PUTUSER Puts the solution from a user array and store it in a SEPRAN solution array. It is supposed that this user array has the structure as defined in COMPSR (18.3.3)

PUTUSBF Exactly the same subroutine as PUTUSER, however, in this case blank common is not used.

COPYCOORUS Copy all coordinates or a selected set of it from the mesh into a user array.

COPYCOORUSBF Exactly the same subroutine as COPYCOORUS, however, in this case blank common is not used.

18.2.1 Subroutine COPYUS

Description

With this subroutine, the user may copy the contents of a SEPRAN solution array or array of special structure, into a user array or vice-versa. Also the user may copy the contents of the solution array in a user point into a user array.
The main reason to use this subroutine is that results from a SEPRAN computation must be used in a user program. In fact the user can always find the solution in specific points using the memory management subroutines of Chapter 22. This subroutine, however, performs the memory management tasks, thus preventing the user of the need to study Chapter 22.

Heading

```
subroutine copyus ( iincop, isol, kmesh, kprob, routcp )
```

Parameters

INTEGER IINCOP(∗), ISOL(5), KMESH(∗), KPROB(∗)

DOUBLE PRECISION ROUTCP(∗)

IINCOP User input array. In this array the user defines which output the user requires from the solution. The output may be restricted to a user point, a curve, a surface or to the complete mesh etc. IINCOP must be filled as follows:

Pos. 1 Number of entries that are filled by the user. All other positions are defined by default values. So if IINCOP(1) = 0 or 1 only defaults are used.

Pos. 2 ICHOICE: Defines what type of nodes must be used.
Possible values:

0: copy contents of isol into ROUTCP.

1: copy contents of ROUTCP into isol. In this case the user must give IPROB (IINCOP(5)) a value. The array ISOL is supposed to be a solution vector.

2: Copy the values of the unknowns in user point IPOINT into ROUTCP.

Default value: 0

Pos. 3 IDEGFD: Defines the specific unknowns in each point that must be copied into ROUTCP. If IDEGFD = 0, all unknowns are copied.
Default value: 0.

Pos. 4 INODE: Defines the user point in which the unknown must be selected (ICHOICE=2).
Default value: 1.

Pos. 5 IPROB: Problem sequence number. This number refers to the problem numbers as introduced in the input of SEPSTR (4.2.1) or PROBDF (4.4). IPROB must only be filled for ichois = 1.
Default value: 1.

Pos. 6 ICOMPL Indication if the vector is real (0) or complex, must only be filled for ichois = 1.
Default value: 0.

ISOL Standard SEPRAN array of length 5, that refers to the solution array or an array of special structure. At this moment only arrays of the type 110 (i.e. solution arrays) are allowed. In the case of one problem ISOL must be a one-dimensional array of length 5 (or larger). In the case of more problems on one mesh ISOL may be declared as a two-dimensional array of length $5 \times n$ (ISOL(5,n)). In the call of COPYUS one can use ISOL(1,$IPROB$)) with IPROB the problem number.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

ROUTCP User array in which the computed value(s) is (are) stored. ROUTCP is a variable length array which implies that the first position must be filled by the user with the declared length. It is advised to set the third position equal to 0. The output is always stored from position 6! If ICHOICE = 1, ROUTCP is an input array to be filled from position 6.
If ISOL corresponds to a complex vector, array ROUTCP must be declared complex $*$ 16, i.e. double precision complex instead of real.

Input

The arrays IINCOP, KMESH and KPROB must have been filled. (IINCOP by the user).
If ICHOICE = 0 or 2, array ISOL must have been filled. If ICHOICE = 1, array ROUTCP must have been filled.

Output

Depending on the value of ICHOICE, either ISOL or ROUTCP has been filled.

18.2.2 Subroutine COPYUSBF

Description

Subroutine COPYUSBF is exactly the same as subroutine COPYUS, in fact it is the body of this subroutine. The only difference is that blank common is not used in this subroutine, but that the arrays IBUFFR and BUFFER are passed through the parameter list.

Heading

```
      subroutine copyusbf ( ibuffr, buffer, iincop, isol, kmesh, kprob,
     +                      routcp )
```

Parameters

INTEGER IINCOP(∗), ISOL(5), KMESH(∗), KPROB(∗), IBUFFR(∗)

DOUBLE PRECISION ROUTCP(∗), BUFFER(∗)

IINCOP, ISOL, KMESH, KPROB, ROUTCP See subroutine COPYUS

IBUFFR Standard integer buffer array as available in blank common.

BUFFER Standard double precision buffer array as available in blank common.

Input

See subroutine COPYUS.

Output

See subroutine COPYUS.

18.2.3 Subroutine PUTUSER

Description

Subroutine putuser is used to put the solution from a user array and store it in a SEPRAN solution array. This subroutine differs from COPYUS in the sense that not only the user array must be given but also the corresponding nodal point numbers through an array INODES. In fact the user array and the array INODES are supposed to be created by subroutine COMPSR (18.3.3) or they must have exactly the structure created by this subroutine.

Heading

```
subroutine putuser ( iincop, isol, kmesh, kprob, funcy, inodes )
```

Parameters

INTEGER IINCOP(∗), ISOL(5), KMESH(∗), KPROB(∗), INODES(∗)

DOUBLE PRECISION FUNCY(∗)

IINCOP Dummy array intended for future uses. In order to guarantee upwards compatibility it is necessary to put a zero in the first position.

ISOL Standard SEPRAN solution array of type 110 or 115. At this moment isol must have been filled or initialized before, for example by subroutine CREATE (5.3.1)

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

INODES Integer array containing the node numbers corresponding to the arrays funcy in the same sequence from position 6.
This must be an array of type 128 as produced by COMPSR (18.3.3).

FUNCY User array of type 117, where the degrees of freedom of the points are stored. FUNCY must be an array of the type produced by COMPCR (18.3.1) or COMPSR (18.3.3).

Input

The arrays ISOL, KMESH, KPROB, FUNCY and INODES must be filled.
Array IINCOP position 1 must have the value 0.

Output

The contents of array FUNCY have been stored in the array corresponding to ISOL.

18.2.4 Subroutine PUTUSBF

Description

Subroutine PUTUSBF is exactly the same as subroutine PUTUSER, in fact it is the body of this subroutine. The only difference is that blank common is not used in this subroutine, but that the arrays IBUFFR and BUFFER are passed through the parameter list.

Heading

```
      subroutine putusbf ( ibuffr, buffer, iincop, isol, kmesh, kprob,
     +                     funcy, inodes )
```

Parameters

INTEGER IINCOP(∗), ISOL(5), KMESH(∗), KPROB(∗), INODES(∗), IBUFFR(∗)

DOUBLE PRECISION FUNCY(∗), BUFFER(∗)

IINCOP, ISOL, KMESH, KPROB, FUNCY, INODES See subroutine PUTUSER

IBUFFR Standard integer buffer array as available in blank common.

BUFFER Standard double precision buffer array as available in blank common.

Input

See subroutine PUTUSER.

Output

See subroutine PUTUSER.

18.2.5 Subroutine COPYCOORUS

Description

With this subroutine, the user may copy the coordinates as referred to in the SEPRAN array KMESH, into a user array. Also a selected set may be copied.
The main reason to use this subroutine is that coordinates from a SEPRAN computation must be used in a user program. In fact the user can always find the coordinates in specific points using the memory management subroutines of Chapter 22. This subroutine, however, performs the memory management tasks, thus preventing the user of the need to study Chapter 22.

Heading

```
subroutine copycoorus ( kmesh, iincop, usercoor )
```

Parameters

INTEGER IINCOP(∗), KMESH(∗)

DOUBLE PRECISION USERCOOR(∗)

IINCOP User input array. In this array the user defines in which nodes the user requires the coordinates. The output may be restricted to a user point, a curve, a surface or to the complete mesh etc. IINCOP must be filled as follows:

Pos. 1 Number of entries that are filled by the user. All other positions are defined by default values. So if IINCOP(1) = 0 or 1 only defaults are used.

Pos. 2 ICHOICE: Defines what type of nodes must be copied.
Possible values:

0: copy all coordinates into USERCOOR.

Default value: 0

KMESH Standard SEPRAN array in which information of the mesh is stored.

USERCOOR User array in which the coordinates are) stored. USERCOOR is a variable length array which implies that the first position must be filled by the user with the declared length. It is advised to set the third position equal to 0. The coordinates are always stored from position 6!
The coordinates are stored in the sequence of the nodal points, first all coordinates of the first point, then of the second and so on.

Input

The arrays IINCOP and KMESH must have been filled. (IINCOP by the user).
The first 5 positions of USERCOOR must have been filled.

Output

Array USERCOOR has been filled.

18.2.6 Subroutine COPYCOORUSBF

Description

Subroutine COPYCOORUSBF is exactly the same as subroutine COPYCOORUS, in fact it is the body of this subroutine. The only difference is that blank common is not used in this subroutine, but that the arrays IBUFFR and BUFFER are passed through the parameter list.

Heading

```
      subroutine copycoorusbf ( ibuffr, buffer, kmesh, iincop,
     +                          usercoor )
```

Parameters

INTEGER IINCOP(∗), KMESH(∗), IBUFFR(∗)

DOUBLE PRECISION USERCOOR(∗), BUFFER(∗)

IINCOP, ISOL, KMESH, KPROB, USERCOOR See subroutine COPYCOORUS

IBUFFR Standard integer buffer array as available in blank common.

BUFFER Standard double precision buffer array as available in blank common.

Input

See subroutine COPYCOORUS.

Output

See subroutine COPYCOORUS.

18.3 Mapping of values along curves (surfaces) into a user array

In this section we describe two subroutines to map values along curves or surfaces into a user array. The following four subroutines are available

COMPCR Mapping of values along curves into a user array.

COMPCRBF Exactly the same subroutine as COMPCR, however, in this case blank common is not used.

COMPSR Mapping of values along surfaces into a user array.

COMPSRBF Exactly the same subroutine as COMPSR, however, in this case blank common is not used.

18.3.1 Subroutine COMPCR

Description

This subroutine computes values of an array along curves indicated by the user and stores these values in the user array FUNCY. The co-ordinates of the nodal points in these curves are stored in the user array FUNCX.

Heading

```
subroutine compcr ( ichoice, kmesh, kprob, ivectr, number, icurvs, funcx, funcy )
```

Parameters

INTEGER ICHOICE, KMESH(∗), KPROB(∗), IVECTR(5), NUMBER, ICURVS(∗)

DOUBLE PRECISION FUNCX(∗), FUNCY(∗)

ICHOICE Choice parameter. Possibilities:

- 0 Standard situation, see description.
- -1 Only the co-ordinates are stored in array FUNCX. In this case the arrays KPROB, IVECTR and FUNCY are not used, nor the parameter NUMBER.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

IVECTR Array containing information of the function to be computed. IVECTR may be for example the solution vector ISOL or the output vector IOUTVC of subroutine DERIV.

NUMBER When NUMBER > 0 then the $NUMBER^{th}$ degree of freedom in each nodal point with respect to the array corresponding to $IVECTR$ is computed.
When NUMBER $= 0$ all degrees of freedom in the nodal points along the curves are computed. NUMBER $= 0$ is only permitted when the number of degrees of freedom is constant in each nodal point, except when the maximal number of degrees of freedom in a point is equal to 1.
When number < 0, one of the following special options is used:

- -1 $\mathbf{u} \cdot \mathbf{n}$ is computed, where the first component of $\mathbf{u}$ is supposed to be the first degree of freedom in the nodal point, the second component, the second degree of freedom etc.

-2 $\mathbf{u} \cdot \mathbf{t}$ is computed, where the first component of $\mathbf{u}$ is supposed to be the first degree of freedom in the nodal point, and the second component, the second degree of freedom.

-3 $\mathbf{n} \cdot \sigma \cdot \mathbf{n}$ is computed, where σ_{11} is supposed to be the first degree of freedom, σ_{12} the second degree of freedom, and σ_{22} the third degree of freedom in the nodal point. (R^2 only).

-4 $\mathbf{n} \cdot \sigma \cdot \mathbf{t}$ is computed, where σ_{11} is supposed to be the first degree of freedom, σ_{12} the second degree of freedom, and σ_{22} the third degree of freedom in the nodal point. (R^2 only).

ICURVS Input array to be defined and declared by the user. The entries of ICURVS must be filled as follows:

1 Indication of how the curves are defined by the user. Possibilities:

0 The function is computed along one curve only. This curve number must be stored in ICURVS(2).
When ICURVS(2)<0 this curve is considered in reversed order.

-1 The function is computed along the curves ICRV1 to ICRV2 in that order, where the first point of ICRV1 is used as initial point.
ICRV1 must be stored in ICURVS(2).
ICRV2 must be stored in ICURVS(3). The curves ICRV1 to ICRV2 must be subsequent curves.
When ICRV2 < ICRV1, the sequence is given by ICRV1, ICRV1-1, . . . , ICRV2.

-2 See ICURVS(1)=-1, however, the curves are used in reversed order. Hence the last point of ICRV1 is used as initial point.

n >0 The function is computed along the curves ICRV1, ICRV2, where the first point of ICRV1 is used as initial point. The curve numbers ICRV1, ICRV2, . . . , ICRVn must be stored in ICURVS(2) , . . . , ICURVS(n+1).
When a negative curve number is given, the curve is used in reversed order.

FUNCX At output the co-ordinates of the nodal points in the curves, where the degrees of freedom to be found are present, are stored in the user array FUNCX from position 6. FUNCX is a variable length array, hence the first position must be filled by the user.

FUNCY At output the degrees of freedom in the curves are stored in the user array FUNCY from position 6. FUNCY is a variable length array, hence the first position must be filled by the user.

Input

ICHOICE and NUMBER must have a value.
The arrays KMESH, KPROB, IVECTR and ICURVS must be filled.

Output

The arrays FUNCX and FUNCY are filled as user arrays of type 117.
Their structure is as follows:

FUNCX

1 declared length (to be filled by the user).

2 117.

3 computed actual length.

4 NDIM: dimension of the space.

5 number of co-ordinates in FUNCX.

6,... co-ordinates of the points in the sequence $x_1, y_1, z_1,\ x_2, y_2, z_2,$
The nodal points are numbered sequentially along the curves indicated.

FUNCY

1 declared length (to be filled by the user).

2 117.

3 computed actual length.

4 number of degrees of freedom per point (= NUNKP when NUMBER=0, and 1 when NUMBER>0).

5 number of degrees of freedom stored in FUNCY.

6,... degrees of freedom of the points, stored in the same sequence as in FUNCX, i.e. $u_1, v_1, w_1,\ u_2, v_2, w_2$.

When the vector corresponding to IVECTR is complex, each degree of freedom is stored in two positions in array FUNCY.

18.3.2 Subroutine COMPCRBF

Description

Subroutine COMPCRBF is exactly the same as subroutine COMPCR, in fact it is the body of this subroutine. The only difference is that blank common is not used in this subroutine, but that the arrays IBUFFR and BUFFER are passed through the parameter list.

Heading

```
      subroutine compcrbf ( ibuffr, buffer, ichoice, kmesh, kprob,
     +                      ivectr, number, icurvs, funcx, funcy )
```

Parameters

INTEGER ICHOICE, KMESH(∗), KPROB(∗), IVECTR(5), NUMBER, ICURVS(∗), IBUFFR(∗)

DOUBLE PRECISION FUNCX(∗), FUNCY(∗), BUFFER(∗)

ICHOICE, KMESH, KPROB, IVECTR, NUMBER, ICURVS, FUNCX, FUNCY See subroutine COMPCR

IBUFFR Standard integer buffer array as available in blank common.

BUFFER Standard double precision buffer array as available in blank common.

Input

See subroutine COMPCR.

Output

See subroutine COMPCR.

18.3.3 Subroutine COMPSR

Description

This subroutine computes values of an array along surfaces indicated by the user and stores these values in the user array FUNCY. The co-ordinates of the nodal points in these surfaces are stored in the user array FUNCX. The nodal point numbers are stored in array INODES.

Heading

```
      subroutine compsr ( ichoice, kmesh, kprob, ivectr, number, isurfs,
     +                    funcx, funcy, inodes )
```

Parameters

INTEGER ICHOICE, KMESH(∗), KPROB(∗), IVECTR(5), NUMBER, ISURFS(∗), INODES(∗)

DOUBLE PRECISION FUNCX(∗), FUNCY(∗)

ICHOICE Choice parameter. Possibilities:

0 Standard situation, see description.

-1 Only the co-ordinates are stored in array FUNCX. In this case the arrays KPROB, IVECTR and FUNCY are not used, nor the parameter NUMBER.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

IVECTR Array containing information of the function to be computed. IVECTR may be for example the solution vector ISOL or the output vector IOUTVC of subroutine DERIV.

NUMBER When NUMBER > 0 then the $NUMBER^{th}$ degree of freedom in each nodal point with respect to the array corresponding to $IVECTR$ is computed.
When NUMBER $= 0$ all degrees of freedom in the nodal points along the surfaces are computed. NUMBER $= 0$ is only permitted when the number of degrees of freedom is constant in each nodal point, except when the maximal number of degrees of freedom in a point is equal to 1.

ISURFS Input array to be defined and declared by the user. The entries of ISURFS must be filled as follows:

1 Number of surfaces (n_surfs) stored in array ISURFS.

2...1+n_surfs surface numbers along which the values must be copied.

FUNCX At output the co-ordinates of the nodal points in the surfaces, where the degrees of freedom to be found are present, are stored in the user array FUNCX from position 6. FUNCX is a variable length array, hence the first position must be filled by the user.
FUNCX has exactly the same structure as in subroutine COMPCR.

FUNCY At output the degrees of freedom in the surfaces are stored in the user array FUNCY from position 6. FUNCY is a variable length array, hence the first position must be filled by the user.
FUNCY has exactly the same structure as in subroutine COMPCR.

INODES At output the nodal point numbers corresponding to the values in FUNCX and FUNCY are stored in the same sequence in this array.

Input

ICHOICE and NUMBER must have a value.
The arrays KMESH, KPROB, IVECTR and ISURFS must be filled.

Output

The arrays FUNCX and FUNCY are filled as user arrays of type 117. See subroutine COMPCR.
Array INODES is filled as user array of type 128. Its structure is as follows:

INODES

1	declared length (to be filled by the user).
2	128.
3	computed actual length.
4	-
5	number of nodes in FUNCX, FUNCY and INODES.
6,...	Node sequence numbers in exactly the same sequence as FUNCX and FUNCY.

18.3.4 Subroutine COMPSRBF

Description

Subroutine COMPSRBF is exactly the same as subroutine COMPSR, in fact it is the body of this subroutine. The only difference is that blank common is not used in this subroutine, but that the arrays IBUFFR and BUFFER are passed through the parameter list.

Heading

```
      subroutine compsrbf ( ibuffr, buffer, ichoice, kmesh, kprob,
     +                      ivectr, number, isurfs, funcx, funcy,
     +                      inodes )
```

Parameters

INTEGER ICHOICE, KMESH(∗), KPROB(∗), IVECTR(5), NUMBER, ISURFS(∗), INODES(∗), IBUFFR(∗)

DOUBLE PRECISION FUNCX(∗), FUNCY(∗), BUFFER(∗)

ICHOICE, KMESH, KPROB, IVECTR, NUMBER, ISURFS, FUNCX, FUNCY, INODES See subroutine COMPSR

IBUFFR Standard integer buffer array as available in blank common.

BUFFER Standard double precision buffer array as available in blank common.

Input

See subroutine COMPSR.

Output

See subroutine COMPSR.

18.4 Intersection of a 2D solution with a line (Subroutine INTP2D)

Description

This subroutine intersects a 2D mesh with a straight line and computes the points in which the sides of the elements intersect the straight line. The input vector (for example the solution vector, or a derivative vector) is interpolated in these intersection points. The computed values are stored in an output vector of type 119 which can be plotted with the aid of subroutine PLOTFN which recognizes arrays of that type.

Call

```
CALL INTP2D ( ICHOICE, IVALUE, VALUES, NUMBER, KMESH, KPROB, ISOL, IOUTVC,
              ISKIP, IHELP )
```

INTEGER ICHOICE, IVALUE, NUMBER, KMESH(∗), KPROB(∗), ISOL(5), IOUTVC(5), ISKIP, IHELP(∗)

DOUBLE PRECISION VALUES(∗)

ICHOICE Choice parameter indicating how the intersection line is defined. Possible values:

1 The user defines a line by defining a starting point and an angle with respect to the positive x-axis under which the line must be drawn in the positive x-direction. The values of the starting point respectively the angle must be stored into the array VALUES as follows:

1 xstart

2 ystart

3 angle of rotation with respect to positive x-axis in degrees.

IVALUE Not yet used

VALUES Double precision array in which the user must store real information concerning the intersection line. For the correct contents see ICHOICE.

NUMBER The $NUMBER^{th}$ unknown in each nodal point in the array referred to by ISOL will be selected. When this unknown does not appear in a node, this point is skipped.
When ISOL corresponds to a complex vector then NUMBER = 1 corresponds to the real part of the first unknown in each nodal point and NUMBER = 2 to the imaginary part. In the same way NUMBER = 3 corresponds to the real part of the second unknown etc. Hence when the real part of the i^{th} unknown is required, NUMBER should have the value $2i$-1, for the imaginary part NUMBER should be equal to $2i$.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition. KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

ISOL Array containing information of the vector to be interpolated. This may be for example the solution vector, but also a vector of related quantities like the derivatives.

IOUTVC Output vector of length 5 containing information of the interpolation. The related interpolation vector contains the x- and f-values which can be plotted by subroutine PLOTFN. IOUTVC is a special array of type 119.

ISKIP Indication if there are element groups that must be skipped in the interpolation procedure (ISKIP=1) or that all element groups must be taken into account (ISKIP=0). By using ISKIP=1, the user is able to skip a part of the mesh and in this way restrict the interpolation to a small part.

IHELP Integer input array of length NELGRP (= number of element groups), that is only used if ISKIP = 1.
If IHELP(IELGRP) = 0 then element group IELGRP is skipped, otherwise
(IHELP(IELGRP) = 1), element group IELGRP is taken into account.

Input

The arrays VALUES, KMESH, KPROB and ISOL must have been filled.
If ISKIP = 1, then array IHELP must have been filled.
The parameters ICHOICE, ISKIP and NUMBER must have a value.

Output

Array IOUTVC has been filled. It has type number 119.

Remark

At this moment it is only possible to plot the output vector with the aid of subroutine PLOTFN.
The print subroutines must still be adapted in order to recognize arrays of type 119.

18.5 Intersection of a 3D solution with a plane (Subroutine INTP3D)

In this section we describe two subroutines that may be used to intersect a three-dimensional region with a plane.
The following four subroutines are available

INTP3D Intersect a 3d region by a plane.

INTP3DBF Exactly the same subroutine as INTP3D, however, in this case blank common is not used.

18.5.1 Subroutine INTP3D

Description

This subroutine intersects a three-dimensional (volume) mesh with a plane. The intersection is represented by a two-dimensional triangular mesh. The corresponding solution stored in ISOL is interpolated onto this triangular linear mesh. The aim of this subroutine is to create three new arrays KMSH2D, KPRB2D and ISOL2D, that can be used to represent the solution in an intersection by using the standard two-dimensional print and plot subroutines.

Heading

```
      subroutine intp3d ( ichoice, values, kmesh, kprob, isol, kmsh2d,
     +                    kprb2d, isol2d, interp )
```

Parameters

INTEGER ICHOICE, KMESH(∗), KPROB(∗), ISOL(5), KMSH2D(∗), KPRB2D(∗), ISOL2D(5), INTERP

DOUBLE PRECISION VALUES(∗)

ICHOICE Choice parameter consisting of three parts: ichoice_plane, ichoice_tang and ichoice_orig according to ichoice = ichoice_plane+ 10 × ichoice_tang+100 × ichoice_orig.
Meaning of the various parts:

ICHOICE_plane Choice parameter indicating how the plane is defined by the user with the aid of array values. Possibilities:

1 Interpolate plane x = xc, with xc stored in values(1).

2 Interpolate plane y = yc, with yc stored in values(1).

3 Interpolate plane z = zc, with zc stored in values(1).

4 Interpolate plane ax + by + cz = d. Array values must be stored with a,b,c and d in that sequence

ICHOICE_orig indicates how the origin must be computed.
Possible values:

0 Computed by the program.

1 Given by the user in values(5 to 7).

ICHOICE_tang indicates how the tangential vectors must be computed, that define the axis in the plane.
Possible values:

0 the tangential vectors are computed automatically.

1 the tangential vectors are defined by the x and y axis, hence the plane must be z = constant

2 the tangential vectors are defined by the y and z axis, hence the plane must be x = constant

3 the tangential vectors are defined by the x and z axis, hence the plane must be y = constant

4 the tangential vectors are defined by the mapping of the x and y axis onto the intersection plane. This is only allowed if the coefficient for z in the plane is unequal to 0.

5 the tangential vectors are defined by the mapping of the y and z axis onto the intersection plane. This is only allowed if the coefficient for x in the plane is unequal to 0.

6 the tangential vectors are defined by the mapping of the x and z axis onto the intersection plane. This is only allowed if the coefficient for x in the plane is unequal to 0.

7 the tangential vectors are given by the user in array values pos. 8-10 and 11-13.

VALUES Double precision array containing information concerning the plane in accordance to ICHOICE.

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

ISOL Array containing information of the vector to be interpolated. This may be for example the solution vector, but also a vector of related quantities like the derivatives.

KMSH2D Array in the same structure as KMESH but containing information of the 2D intersection (triangles only).

KPRB2D Array in the same structure as KPROB but containing information of the 2D intersection.

ISOL2D Array in the same structure as ISOL but containing information of the 2D intersection (interpolated values).

INTERP Indication of what variables must be interpolated.
INTERP consists of three parts NUNKP, NUMBER and ITRANS according to
INTERP = NUNKP + 100 × NUMBER + 10000 × ITRANS, with

NUMBER indicates which is the first degree of freedom that must be interpolated in each point. When NUMBER = 0 is given, it is interpreted as NUMBER = 1

NUNKP indicates the number of unknowns that must be interpolated in each point containing this number of unknowns counted from NUMBER.
The resulting array ISOL2D corresponds to an array with a fixed (nunkp) number of unknowns in each point. Hence when an unknown with number in the range NUMBER to NUMBER+NUNKP-1 does not exists, this point is not used in the output. It is necessary that the unknowns to be interpolated are at least present in vertex nodal points

ITRANS Indicates what type of transformation must be applied if NUMBER ≥ 3. The following values for ITRANS are available

0 The output vector is not transformed. So the components of the output vector remain in same the direction as before the intersection.

1 The first 3 components of the output vector are transformed such that the third component is perpendicular to the intersection plane and the first two components are orthogonal components within the plane. All other components are not transformed.

Input

The arrays VALUES, KMESH, KPROB and ISOL must have been filled.
The parameters ICHOICE and INTERP must have a value.

Output

The arrays KMSH2D, KPRB2D and ISOL2D have been filled. They may be used in all standard SEPRAN postprocessing subroutines.

18.5.2 Subroutine INTP3DBF

Subroutine INTP3DBF is exactly the same as subroutine INTP3D, in fact it is the body of this subroutine. The only difference is that blank common is not used in this subroutine, but that the arrays IBUFFR and BUFFER are passed through the parameter list.

Heading

```
 subroutine intp3dbf ( ibuffr, buffer, ichoice, values, kmesh,
+                      kprob, isol, kmsh2d, kprb2d, isol2d,
+                      interp )
```

Parameters

INTEGER ICHOICE, KMESH(∗), KPROB(∗), ISOL(5), KMSH2D(∗), KPRB2D(∗), ISOL2D(5), INTERP

DOUBLE PRECISION VALUES(∗) See subroutine INTP3D

IBUFFR Standard integer buffer array as available in blank common.

BUFFER Standard double precision buffer array as available in blank common.

Input

See subroutine INTP3D.

Output

See subroutine INTP3D.

18.6 Mapping from polar to Cartesian co-ordinates (Subroutine TRANPL)

Description

This subroutine may be used to transform both the co-ordinates and the solution corresponding to a mesh, from polar to cartesian co-ordinates. This solution vector must be a vector consisting of two unknowns per point. These components should correspond to a polar co-ordinate system, and are transformed to a cartesian form. After the call of this subroutine the original solution array as well as the original co-ordinates have been lost.

Call

```
CALL TRANPL ( ICHOICE, KMESH, KPROB, ISOL )
```

INTEGER ICHOICE, KMESH(∗), KPROB(∗), ISOL(5)

ICHOICE Not yet in use.

KMESH Standard SEPRAN array in which information of the mesh is stored.
At input the co-ordinates are assumed to be defined in polar form, at output the co-ordinates are transformed to cartesian form.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

ISOL Array containing information of the solution vector to be transformed. At input the solution vector is assumed to be vector with two polar components, at output the components are transformed to cartesian form.

Input

The arrays KMESH, KPROB and ISOL must have been filled.

Output

The contents of the arrays corresponding to KMESH (co-ordinates) and ISOL (solution array) have been changed.

18.7 Mapping of one matrix structure into another one (Subroutine MAPCOM)

Description

This subroutine maps depending on the choice parameter ICHOICE in IINMAP either array INTMT1 onto array INTMT2 or array MATR1 onto MATR2, where INTMT1 and INTMT2 correspond to different storage methods.
The arrays INTMT1 and INTMT2 must correspond to the same problem.
Restriction
At this moment INTMT1 must correspond to JMETHOD = 5, and INTMT2 to JMETHOD = 9 or if ICHOICE = 2, INTMT1 corresponding to JMETHOD = 1, and INTMT2 corresponding to JMETHOD = 5 is also permitted.

Call

```
CALL MAPCOM ( IINMAP, KPROB, INTMT1, INTMT2, MATR1, MATR2 )
```

INTEGER IINMAP(∗), KPROB(∗), INTMT1(5), INTMT2(5), MATR1(5), MATR2(5)

IINMAP Integer input array in which the user defines the actions to be taken by this subroutine. IINMAP must be filled as follows:

1 Highest entry number of the array that has been filled by the user. The user must fill all positions 1 to IINMAP(1) in array IINMAP. For all other positions default values are used. Hence, if IINMAP(1)=0 or 1 only defaults are used.

2 ICHOICE, defines the type of action to be taken. Possible values:

1 Map INTMT1 onto INTMT2

2 Map MATR1 onto MATR2

Default value: 1

KPROB Standard SEPRAN array, containing information of the problem definition.

INTMT1 Standard SEPRAN array containing information of the structure of the matrix. This structure is translated to the new structure defined in INTMT2.

INTMT2 Standard SEPRAN array containing information of the structure of the second matrix. If ICHOICE = 1 this structure is translated from the old structure defined in INTMT1. In that case INTMT2 must have been partially filled before:

1 JMETHOD + 1000 × (IPROB-1), with IPROB the problem number and JMETHOD the storage sequence number as defined in the input block MATRIX or subroutine MATSTRUC (4.5)

2 102

3 0 or memory management sequence number corresponding to preceding first part

4 0 or memory management sequence number corresponding to preceding second part

5 -

If ICHOICE=2 INTMT2 must have been filled completely

MATR1 Standard SEPRAN array containing information of the matrix to be copied (ICHOICE=2 only)

MATR2 Standard SEPRAN array containing information of the copied matrix (ICHOICE=2 only)

Input

The arrays IINMAP, KPROB and INTMT1 must have been filled.
Array INTMT2 must have been (partly) filled.
Depending on ICHOICE array MATR1 must have been filled.

Output

Array INTMT2 has been filled as well as the part in IBUFFR corresponding to it.
Depending on ICHOICE array MATR2 has been filled.

19 Auxiliary subroutines

19.1 Introduction

In this chapter we treat a number of auxiliary subroutines that do not naturally belong to other chapters.
The following items are considered:

19.2 Timing subroutines. This concerns some special machine-dependent subroutines that may be used to compute and print CPU times.
The call of the subroutines is machine-independent.

19.3 Opening and closing of files. These are also special machine-dependent subroutines that may be used to open and close SEPRAN files.
The call of the subroutines is machine-independent.

19.4 Subroutines to read and interpret a user input file.

19.5 Special user subroutines. It handles here some user written subroutines that do not naturally belong to other chapters.
At this moment it concerns

19.5.1 Subroutine USEROUT to perform output or to manipulate solutions within program SEPCOMP.

19.2 Timing subroutines

SEPRAN contains two subroutines that may be used to print the CPU-time:

INICLK (19.2.1) The call to this subroutine set the timer back to zero.
This subroutine is called by the SEPRAN starting subroutine.

ASKCLK (19.2.2) Prints the accumulated CPU time since the last call of INICLK.

19.2.1 Subroutine INICLK

Description

Sets the timer equal to zero.
INICLK is called by the starting subroutine.
INICLK is a machine-dependent subroutine.

Call

```
CALL INICLK
```

Output

The timer is reset to zero.

19.2.2 Subroutine ASKCLK

Description

The actual CPU time in seconds since the preceding call of INICLK (19.2.1) is printed.
ASKCLK is called by the main SEPRAN subroutines depending on the parameter ITIME in common CINOUT.
ASKCLK is a machine-dependent subroutine.

Call

```
CALL ASKCLK ( NAME )
```

Parameters

CHARACTER $*$ **10** NAME

NAME Name of calling subroutine.

Input

Name must have a value.

Output

Print of accumulated CPU time.

19.3 Opening and closing of files

In this section we treat the subroutines that are available to open and close files in SEPRAN.
The following subroutines are available:

FINISH (19.3.1) Close all the open SEPRAN files.

OPENF2 (19.3.2) Open the sepran backing storage file *sepranback*

INIFIL (19.3.3) Machine dependent subroutine that may be used to open standard SEPRAN files. Subroutine INIFIL is called by for example the SEPRAN starting subroutines.

19.3.1 Subroutine FINISH

Description

Closes the standard SEPRAN files provided they have been opened before.
FINISH must be called at the end of a SEPRAN program.
Depending on the value of ISTOP, the administration of file 2 is written to backing storage.

Call

```
CALL FINISH ( ISTOP )
```

Parameters

INTEGER ISTOP

ISTOP Choice parameter with the following possibilities:

0 No action with respect to file 2 is taken.

1 The administration of file 2 is written from main memory to file.

Input

ISTOP must have a value.

Output

All SEPRAN files have been closed.

19.3.2 Subroutine OPENF2

Description

Open the sepran backing storage file *sepranback* (file 2) and read preceding contents if necessary

. *Call*

```
CALL OPENF2 ( NEW )
```

Parameters

LOGICAL NEW

NEW Indication if an old File 2 must be used, and hence the information of this file must be read (false) or that the File is considered as new, which means that existing information is destroyed (true)

Input

NEW must have a value.

Output

The sepran backing storage file has been opened.

19.3.3 Subroutine INIFIL

Description

The task of INIFIL is to open all standard SEPRAN 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.

Call

```
CALL INIFIL ( IFILE )
```

Parameters

INTEGER IFILE

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

1	Standard element input file (iref1)
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 mesh (iref10) (meshoutput)
17	Standard menu message file (iref17)
73	File containing information of the problem (iref73) (sepcomp.inf)
74	File containing the solutions (iref74) (sepcomp.out)

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 the error parameter ierror in common block CCONST has the value -1, ierror is set to 1 and control is given back to the calling subroutine , otherwise the program is stopped.
With respect to the files 1 4 and 17 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.
Depending on the value of IFILE, the file to be opened, must exist.

Output

The file indicated by the parameter IFILE has been opened.

19.4 Help subroutines to read user input from the input file

In this section we consider some of the subroutines that SEPRAN uses to interpret the user input file. These subroutines may also be used by the user himself to read his own input file and to use the idea of keywords and free format input as present in SEPRAN.
The subroutines that are used mostly are:

ASKNXT (19.4.1) Reads next item from the input file and returns with the value of this item and the type of item found. Also end of line and end of input file are noticed.

INTX00 (19.4.2) Checks if a keyword read by ASKNXT belongs to a given table. It returns with the position of the keyword in that table.

INTX01 (19.4.3) Has exactly the same task as INTX00, however, the number of significant characters in the keyword is constant for all items in the table, whereas this number is variable in INTX00.

INTX02 (19.4.4) Is the most general reading subroutine that SEPRAN contains. In fact use of INTX02 is sufficient in nearly all cases. INTX02 does not need the combination with one of the other subroutines treated in this chapter. INTX02 itself uses subroutine ASKNXT.
INTX02 is able to read a part of the SEPRAN input file and to return with the items read in a form that can be manipulated easily.

INTX04 (19.4.5) is a special subroutine to read a series of numbers in the SEPRAN input file.

19.4.1 Subroutine ASKNXT

Description

Subroutine ASKNXT is used to read the next item (keyword, string or number) from the input file. It is assumed that the input file has already been opened. The input file is defined as the file coupled to the file reference number IREFWR as stored in common block CMCDPI (21.5). Usually this is the standard input file but the user may change this file number locally, provided he takes care of resetting the value if necessary.

Heading

```
subroutine asknxt
```

Parameters

Subroutine ASKNXT does not have any parameters. However, the communication is completely defined by two common blocks CSEPRD and CSEPCH. These common blocks are defined as follows:

```
      double precision realrd
      integer nchar, ityprd, icolum, kcolum, irecnr, ispecl, istop,
     +        lastp, last
      common /cseprd/ realrd, nchar, ityprd, icolum, kcolum, irecnr,
     +                ispecl, istop, lastp, last
      save /cseprd/
```

Common block CSEPRD contains all real and integer information that is needed by subroutine ASKNXT. The parameters, which are initialized by the SEPRAN starting subroutines, have the following meaning:

REALRD Last real or integer number that has been read from the input file

NCHAR Number of characters found in the string.

ITYPRD Indication of what kind action has been taken place last.
Possible values:

0 End of file found.

1 End of record found.

2 integer has been read.

3 Real has been read.

4 Keyword has been read.

5 Text between quotes has been read or text in the case of ispec=3.

6 - sign not followed by digit (ispecl=4 only).

7) sign (ispecl=4 only).

8 (sign (ispecl=5 only).

9 = sign (ispecl=6 only).

ICOLUM Last column number that has been used.

KCOLUM Last column number that could be interpreted.

IRECNR Record sequence number.

ISPECL Parameter indicating if the input must be treated in the standard way or that special rules must be applied.
Possible values:

0 Normal case.

1 Special situation for the coarse a111, not treated here.

2 Special situation for the coarse a111, not treated here.

3 Characters are not translated to upper case. The text is interpreted as a string.

4 Special case, the program also finds the occurrences of the - sign not followed by a digit and the) or (sign.

5 Special case, the occurrence of a (sign is marked.

6 Special case: the environment file is read, so error messages can not be read from the error file.

7 Combination of 3 and 6.

8 Special case, the next equals sign is found and the position following it is returned in kcolum.

9 Special case, a keyword may also contain digits and underscores. It must, however, start with a letter.

ISTOP Indicates which actions must be taken if end of file or end of record has been found.
Possible values:

0 Stop the process if end of file has been found.

1 Return with ityprd=0 if end of file has been found.

2 Read next record if end of record has been found, return with ityprd=0 if end of file has been found.

LASTP Position of last non-blank in record.

LAST Position of last non-blank in record before the comment sign `#`.

```
character * 80 string, cards, cardim
common /csepch/ cards, string, cardim
save /csepch/
```

Common block CSEPCH contains all character information that is needed by subroutine ASKNXT. The parameters, which are initialized by the SEPRAN starting subroutines, have the following meaning:

CARDS Contains the last record read in character format.
This record has been manipulated slightly. For example all lower case letters are capitalised except if ISPECL=3. Furthermore the position of the characters may have been moved in the record.

STRING Contains the last string that has been read.

CARDIM Contains the record read in original form.

Input

The parameters ITYPRD, ICOLUM, IRECNR, ISPECL, ISTOP, LASTP and LAST in common block CSEPRD must have a value.

Output

The parameters in the common blocks CSEPRD and CSEPCH have been updated.

19.4.2 Subroutine INTX00

Description

Subroutine INTX00 may be used to check whether a text read by ASKNXT and stored in common CSEPCH, belongs to the table CODE. If this text belongs to CODE INTX00 returns with the sequence number of this text in CODE, otherwise INTX00 gets the value 0. The number of characters that must be compared is stored in the table NCODE.

Heading

```
function intx00 ( code, ncode, n, ierpri, ncodmx )
```

Parameters

integer intx00, n, ncode(n), ierpri, ncodmx(n)

character ∗(∗) code(n)

intx00 Gives the sequence number with respect to the table CODE of the text STRING in common block CSEPCH (19.4.1).
So if string(1:ncode(i)) = code(1:ncode(i))(i), intx00 gets the value i.
If none of the texts match, intx00 gets the value 0.

code Table of length N that must contain the texts to be compared with the text stored in the parameter STRING in common block CSEPCH (19.4.1).

ncode Number of characters that must be compared.
So string(1:ncode(i)) is compared with code(1:ncode(i))(i).

n Number of texts that have been stored in CODE.

ierpri Indicates if an error message must be printed if the text stored in STRING can not be found CODE (1) or not (0).

ncodmx Table containing the maximum number of characters in the keyword table code. If the keyword contains more characters a warning is given.

Input

The arrays CODE, NCODE and NCODMX must have been filled.
The parameters N and IERPRI must have a value.

Output

INTX00 returns with the sequence number of the keyword in the table CODE, or if the keyword can not be found with the value 0.

19.4.3 Subroutine INTX01

Description

Subroutine INTX01 may be used to check whether a text read by ASKNXT (19.4.1) and stored in common CSEPCH (19.4.1), belongs to the table CODE.
If this text belongs to CODE, INTX01 returns with the sequence number of this text in CODE, otherwise INTX01 gets the value 0.
The number of characters that must be compared is stored in the integer NCODE.
The difference with subroutine INTX00 (19.4.2), is that all texts are compared with the same number of characters.

Heading

```
function intx01 ( code, ncode, n, ierpri )
```

Parameters

integer intx01, n, ncode, ierpri

character ∗**(**∗**)** code(n)

intx01 Gives the sequence number with respect to the table CODE of the text STRING in common block CSEPCH (19.4.1).
So if string(1:ncode) = code(1:ncode)(i), intx01 gets the value i.
If none of the texts match, intx01 gets the value 0.

code Table of length N that must contain the texts to be compared with the text stored in the parameter STRING in common block CSEPCH (19.4.1).

ncode Number of characters that must be compared.
So string(1:ncode) is compared with code(1:ncode)(i).

n Number of texts that have been stored in CODE.

ierpri Indicates if an error message must be printed if the text stored in STRING can not be found CODE (1) or not (0).

Input

Array CODE must have been filled.
The parameters N, NCODE and IERPRI must have a value.

Output

INTX01 returns with the sequence number of the keyword in the table CODE, or if the keyword can not be found with the value 0.

19.4.4 Subroutine INTX02

Description

Depending on the parameters ITYPRD and ISTOP in common block CSEPRD (19.4.1), INTX02 reads zero, one or more records. The present record is interpreted and if more records may be read these records are interpreted as well.
In the standard case INTX02 assumes that a record has already been read, for example by ASKNXT or a previous call of INTX02. If, however, at input ITYPRD has the value 1, INTX02 starts by reading the next record itself.
If ISTOP has the value 0 or 1 only one record is read. If ISTOP has the value 2 next records are also read until either a main keyword is found or end of file has been detected.
All keywords read are compared with the keywords stored in the table CODE. Several possibilities are detected, like keyword followed by keyword, or keyword followed by number.
It is checked if values read are within a given range.
If an item is not recognized an error message is given.
If a main keyword is found or end of record the subroutine returns.

Heading

```
subroutine intx02 ( code, ncode, n, itype, table, codemn, nmcode,
                    nmain, iretur, tabmin, tabmax, tabout, nout,
                    strout, nstrng, ncodmx )
```

Parameters

integer n, ncode(n), itype(∗), nmain, nmcode(∗), iretur, nout, nstrng, ncodmx(n)

character ∗**(**∗**)** code(n), codemn(∗), strout(nstrng)

double precision table(n), tabmin(n), tabmax(n), tabout(nout)

code Table of length N containing the keywords that are recognized by INTX02.

ncode Table containing the number of significant characters in the keyword table code. Hence the first NCODE(i) characters of keyword CODE(i) are significant.

n Number of texts stored in code.

itype Table containing information about the items that are expected behind the keywords in the table code. TABLE(i) corresponds of course to CODE(i). Here we give a short description of the input values of ITYPE(i) that are allowed. For an extended description, see *extended description of input*.

-1 Skip the corresponding keyword.
0 keyword = integer.
1 keyword = real.
2 keyword, Set table to 1.
3 keyword or keyword = integer.
4 keyword or keyword = integer, indicate both situations.
5 keyword = keyword.
6 keyword = 'text'.
10 keyword iseq1 = j1, keyword iseq2 = j2, ...
11 keyword = i1, i2, i3, ... all positive integer.
12 keyword iseq1 = j1, keyword iseq2 = j2, ... or keyword = j1 (iseq1 is set to 1).
13 keyword = keyword1 [keyword2], the last may be empty.

14 c1, -c3, c4, -c9, c5, i.e. list of keywords followed by sequence number.

15 c1 v1 c2 v2 List of keyword followed by sequence number followed by another keyword followed by sequence number and this repeatedly.

16 keyword = real or
keyword (value = real) or
keyword (func = i) or
keyword (time_func = i).

17 keyword = ((x1,y1,..), (x2,y2,..) ... (xn,yn,..))

18 keyword = P1, P2, ...

19 keyword = (x1, y1, z1)

20 keyword or keyword keyword = i1, i2, ...

21 keyword (ax+by+cz=d)

22 keyword [keyword1] [keyword2] = (x1, y1, z1)

23 keyword keyword set table to 1

24 keyword [keyword1] [keyword2] = (i1, i2, i3)

25 keyword iseq1 = real or
keyword iseq1 (value = real) or
keyword iseq1 (func = i) or
keyword iseq1 (time_func = i)

26 keyword = complex or
keyword (value = complex) or
keyword (func = i) or
keyword (time_func = i)
with complex = real or (real,real)

27 keyword = series of: real or
keyword (value = real) or
keyword (func = i) or
keyword (time_func = i)

28 keyword = i1, i2, i3, ... all integer

29 keyword iseq = [keyword1] i1, i2, i3, ... all integer

30 keyword iseq1 = c1, keyword iseq2 = c2, ...

codemn Table of length at least NMAIN containing the main keywords that can be recognized by INTX02. The first NMAIN keywords are main keywords. It is possible that there are also subsubkeywords stored in CODEMN. See array ITYPE.

nmcode Table containing the number of significant characters in the main keyword table CODEMN, including those of the extra subsubkeywords. Hence the number of significant characters in keyword CODEMN(i) is equal to NMCODE(i).

nmain $|NMAIN|$ defines the number of main keywords that are stored in CODEMN. CODEMN itself may be larger because subsubkeywords are also stored in CODEMN. If NMAIN < 0 and a main keyword has been found, ICOLUM in common block CSEPRD (19.4.1) is reset to the preceding value, which implies that the main keyword may be read again in the next call of one of the subroutines mentioned in this section.

iretur Return code. Possible values:

-10 At input means that no error message is given if a single real or integer is found. The corresponding value is stored in tabout, next position.
iretur gets the value -10-number of numbers found.

-2 End of file has been reached.

-1 An error has been detected.

0 Normal end of subroutine.

> 0 Main keyword with sequence number IRETUR has been found.

table Contains the numbers that are read for the various keywords. What will be read and stored strongly depends on ITYPE.
At input TABLE must be filled with the default values.

tabmin Must contain the minimum values of the numbers read. The exact contents depends on ITYPE.

tabmax Must contain the maximum values of the numbers read. The exact contents depends on ITYPE.

tabout Contains series of numbers, depending on ITYPE.

nout length of array TABOUT.

strout Is used to store a series of strings, depending on ITYPE and the input file.

nstrng Length of array STROUT.

ncodmx Table containing the maximum number of characters in the keyword table code. If the keyword contains more characters a warning is given. So the maximum number of characters expected in keyword CODE(i) is NCODMX(i).

Input

The arrays CODE, CODEMN, ITYPE, NCODE, NCODMX, NMCODE, TABLE, TABMIN and TABMAX must have been filled.
The parameters IRETUR, N, NMAIN, NOUT, NSTRNG must have a value.

Output

The parameter IRETUR has got a value.
The arrays STROUT, TABLE and TABOUT have been filled by the subroutine.

Extended description of input

The input and output of this subroutine strongly depends on the values of ITYPE(i). ITYPE(i), i = 1 (1) N, defines how keyword CODE(i) must be interpreted and what output is produced. The following values of ITYPE(i) are allowed:

-1 Skip the corresponding keyword. This possibility is meant to introduce keywords that are only meant to increase the readability.
Input:

CODE(i) The name of the keyword.

Output

none

0 The input expected is of the shape `keyword = integer value`.
Input:

CODE(i) The name of the keyword.

TABLE(i) Default value for the integer. If no default value is available, some specific value to indicate the difference between the case that the keyword is present or not. A typical value might be -1d19, or, -1d0, if the only values to be expected are positive.

TABMIN(i) Minimum value allowed for the integer.

TABMAX(i) Maximum value allowed for the integer.

Output

TABLE(i) Integer value read. If the keyword is not present the input value of TABLE(i) is kept.

1 The input expected is of the shape `keyword = real value`.
Input:

CODE(i) The name of the keyword.

TABLE(i) Default value for the real. If no default value is available, some specific value to indicate the difference between the case that the keyword is present or not. A typical value might be -1d19, or, -1d0, if the only values to be expected are positive.

TABMIN(i) Minimum value allowed for the real.

TABMAX(i) Maximum value allowed for the real.

Output

TABLE(i) Real value read. If the keyword is not present the input value of TABLE(i) is kept.

2 The input read is the keyword only. So in this case the appearance of a specific keyword, with no other information, is detected.
Input:

CODE(i) The name of the keyword.

TABLE(i) Any value except 1. It is advised to use a negative value.

Output

TABLE(i) If the keyword is found TABLE(i) gets the value 1, otherwise the input value is kept.

3 The input expected is either a stand alone keyword or `keyword = integer value`.
Input:

CODE(i) The name of the keyword.

TABLE(i) Default value for the integer. If no default value is available, some specific value to indicate the difference between the case that the keyword is present or not. A typical value might be -1d19, or, -1d0, if the only values to be expected are positive.

TABMIN(i) Minimum value allowed for the integer.

TABMAX(i) Maximum value allowed for the integer.

Output

TABLE(i) Integer value read. If the keyword is not present or if the keyword is not followed by an integer value, the input value of TABLE(i) is kept.

4 See ITYPE(i)=3. The difference with ITYPE(i)=3 is that in this case, it is indicated if the keyword has been found in the input file, and if it has been found whether it is followed by an integer value or not.
Input:

CODE(i) The name of the keyword.

TABLE(i) Some value that is not produced as output for example -1d19.

TABMIN(i) Minimum value allowed for the integer.

TABMAX(i) Maximum value allowed for the integer.

Output

TABLE(i) If the keyword has been found without integer number then TABLE(i) gets the value -1000.
If the keyword is followed by an integer,TABLE(i) gets the value IPOSUT (> 0). See TABOUT.
If the keyword is not present in the input file TABLE(i) is let unchanged.

TABOUT(IPOSUT) gets the value of the integer read, if the keyword is followed by an integer value.

5 The input expected is of the shape `keyword = keyword`.
Hence the keyword is followed by a second keyword possibly separated by some separation character like `=`.
Input:

CODE(i) The name of the keyword.

TABLE(i) Some non-positive value.

TABMIN(i) Starting position of the names of the second keyword in the table CODEMN.

TABMAX(i) End position of the names of the second keyword in the table CODEMN.

CODEMN positions TABMIN(i) to TABMAX(i) Names of the second keyword that are allowed.

Output

TABLE(i) If the keyword has been found then TABLE(i) contains the relative position of the second keyword in CODEMN. For example if CODEMN(4) to CODEMN(6) contain the possible second keywords and the second one is found, i.e. the one stored in CODEMN(5), then TABLE(i) gets the value 2.
If the first keyword is not found, TABLE(i) is left unchanged.

6 The input expected is of the shape `keyword = 'text'`. Hence the keyword must be followed by some text between quotes.
Input:

CODE(i) The name of the keyword.

TABLE(i) Some non-positive value.

Output

TABLE(i) If the keyword has been found then TABLE(i) gets the value KPOSUT (> 0), see STROUT.
If the keyword is not found, TABLE(i) is left unchanged.

STROUT(KPOSUT) contains the text between the quotes. The maximum length is implicitly defined by the input parameters.

10 The input expected is of the shape `keyword iseq1 = j1, keyword iseq2 = j2, ...`
This is a special situation of a keyword followed by a sequence number followed by a number, for example `inpvc i = j`.
In this case the sequence numbers iseq1, iseq2, ... must be positive.
The positions i to $i + max - 1$, where max the maximum range of the sequence numbers of the arrays CODE, ITYPE, TABMIN and TABMAX are reserved for this single keyword.
Input:

CODE(i to $i + max - 1$) The name of the keyword, max times repeated.

TABLE(i to $i + max - 1$) Some values that are outside the range of the numbers `j1, j2, ....`

TABMIN(i) Minimum value of the sequence numbers.

TABMAX(i) Maximum value of the sequence numbers. Hence max = TABMAX(i)-TABMIN(i)+1.

TABMIN($i + 1$) Minimum value of the numbers `j1, j2, ....`

TABMAX($i + 1$) Maximum value of the numbers `j1, j2, ....`

ITYPE($i + 1$) indicates if the numbers `j1, j2, ...` are all integer (0) or not (1). In the last case they are al treated as if they are real.

Output

TABLE(i to $i + max - 1$) contain the values of the numbers `j1, j2, ..` at the positions corresponding to the sequence numbers. If a sequence number is not found in the input file, the input value of TABLE(i+iseq-1) is kept.

11 The input expected is of the shape `keyword = i1, i2, i3, ...`, all positive integer.
Special situation of keyword followed by series of positive integer numbers
. Brackets are treated as separator.
Input:

CODE(i) The name of the keyword.

TABLE(i) Some non-positive value.

TABMIN(i) Minimum value of the integers.

TABMAX(i) Maximum value of the integers.

Output

TABLE(i) If the keyword has been found then TABLE(i) gets the value IPOSUT (> 0), see TABOUT.
If the keyword is not found, TABLE(i) is left unchanged.

TABMAX(i) m, the number of integers that have been read.

TABOUT(IPOSUT) to TABOUT(IPOSUT+m-1 contain the values of the integers that have been read.

12 The input expected is of the shape:

```
keyword iseq1 = j1, keyword iseq2 = j2, ...
   or
keyword = j1      (iseq1 is set to 1).
```

Special situation of keyword followed by sequence number followed by number, for example `inpvc i = j` or alternatively keyword followed by number for example `inpvc = j`. i = 1 in that case.
So in fact this case is identical to ITYPE(i) = 10, however, it is not necessary that `i` is present in the input file.
Input:

CODE(i to $i + max - 1$) The name of the keyword, max times repeated.

TABLE(i to $i + max - 1$) Some values that are outside the range of the numbers `j1, j2, ....`

TABMIN(i) Minimum value of the sequence numbers.

TABMAX(i) Maximum value of the sequence numbers. Hence max = TABMAX(i)-TABMIN(i)+1.

TABMIN($i + 1$) Minimum value of the numbers `j1, j2, ....`

TABMAX($i + 1$) Maximum value of the numbers `j1, j2, ....`

ITYPE($i+1$) indicates if the numbers `j1, j2, ...` are all integer (0) or not (1). In the last case they are al treated as if they are real.

Output

TABLE(i to $i + max - 1$) contain the values of the numbers `j1, j2, ..` at the positions corresponding to the sequence numbers. If a sequence number is not found in the input file, the input value of TABLE(i+iseq-1) is kept.

13 The input expected is of the shape `keyword = keyword1 [keyword2]`, the last may be empty.
special situation of keyword followed by one or two keywords, for example: `renumber = cuthill profile`.
Input:

CODE(i) The name of the keyword.

CODE(i+1) The same keyword.

TABLE(i) Some non-positive value.

ITYPE(i) IPOSTYP, i.e. position in array ITYPE. IPOSTYP must be larger than N. IPOSTYP gives the starting address of information concerning the set of keywords that may be used as keyword2.

TABMIN(i) IPMN1.
The set of keywords corresponding to keyword 1 is stored in array CODEMN from position IPMN1.

TABMAX(i)) IPMN2.
The set of keywords corresponding to keyword 1 is stored in array CODEMN from position IPMN1 until position IPMN2. Hence there are M = IPMN2 - IPMN1 + 1 possible keywords for keyword 1.
For each of these keywords a separate set keyword2 is allowed.

ITYPE(IPOSTYP+2×j)-2 (j=1, 2 ... , M) IPMNj1.
The j^{th} set of keywords corresponding to keyword 2 is stored in array CODEMN from position IPMNj1.

ITYPE(IPOSTYP+2×j11) (j=1, 2 ... , M) IPMNj2.
The j^{th} set of keywords corresponding to keyword 2 is stored in array CODEMN from position IPMNj1 until position IPMNj2.

CODEMN(IPMN1) to CODEMN(IPMN2) List of all keywords `keyword1` allowed.

CODEMN(IPMNj1) to CODEMN(IPMNj2) List of all keywords `keyword2` allowed corresponding to the j^{th} keyword1.

Output

TABLE(i) Relative sequence number of keyword1 in list of keywords allowed. Hence if the second keyword of the list of keyword1 is found TABLE(i) gets the value 2.

TABLE(i+1) Relative sequence number of keyword2 in list of keywords allowed. Hence if the second keyword of the list of keyword2 corresponding to keyword1 is found TABLE(i+1) gets the value 2.
If the keyword is not found, TABLE(i) is left unchanged.

14 The input expected is of the shape `c1, -c3, c4, -c9, c5`, i.e. list of keywords followed by sequence number.
Special situation of keyword followed by number, followed by same keyword followed by number and so on, for example: `p1, p3, p5, p8, p2`. The keywords may be preceded by a minus sign, hence `c1, c3, -c5, c8` etc., in which the sequence number gets the minus sign.
Input:

CODE(i) The name of the keyword.

TABLE(i) Some non-positive value.

TABMIN(i) Maximum number of integers that are allowed.

TABMAX(i) Maximum value of the integers that are read.

Output

TABLE(i) If the keyword has been found then TABLE(i) gets the value IPOSUT (> 0), see TABOUT.
If the keyword is not found, TABLE(i) is left unchanged.

TABMAX(i) m, i.e. the number of integers read.

TABOUT(IPOSUT) to TABOUT(IPOSUT+m-1) contain the integer values read.

15 The input expected is of the shape `c1 v1 c2 v2` List of keyword followed by sequence number followed by another keyword followed by sequence number and this repeatedly.
Special situation of keyword followed by number, followed by same keyword followed by number etc. However, in this case it is possible that the keyword number combination is followed by other keywords, before it is followed by the next keyword number combination.
Example: `c1 v1 + c2 v2`.
In this case both c and v get type number 15. In this case in TABOUT exactly

TABMIN(i) positions are reserved. The rest of the input is exactly the same as for ITYPE(i) = 14

16 The input expected is of one of the following shapes

```
keyword = real
keyword (value = real)
keyword (func = i)
keyword (time_func = i)
```

Input:

CODE(i) The name of the keyword.

TABLE(i) Some value that can not occur as output.

Output

TABLE(i) If the keyword has been found then TABLE(i) gets the value IFUNC (> 0), or IPOSUT (< 0), see TABOUT. If a real value is found then this value is stored in TABOUT(-IPOSUT), if `func = j` is found TABLE(i) get the value `j`.
If `time_func = j` is found TABLE(i) get the value -100-`j`.
If the keyword is not found, TABLE(i) is left unchanged.

TABOUT(-IPOSUT) gets the real value read.

17 The input expected is of the shape `keyword = ( (x1,y1,..), (x2,y2,..) ... (xn,yn,..) )`
Special situation of keyword followed by series of series of numbers between brackets. Each series of local numbers itself must be between brackets. The number of numbers in the inner series must be constant.
Input:

CODE(i) The name of the keyword.

TABMIN(i) Number of numbers in the inner series.

TABMAX(i) Maximum number of series.

TABLE(i) Some non-positive value.

Output

TABLE(i) If the keyword has been found then TABLE(i) gets the value IPOSUT (> 0), see TABOUT.
If the keyword is not found, TABLE(i) is left unchanged.

TABMAX(i) m, number of numbers read, i.e. TABMIN(i) $\times$ number of series read.

TABOUT(IPOSUT) to TABOUT(IPOSUT+m-1) contain the numbers read.

18 The input expected is of the shape `keyword = P1, P2, ....`
Special situation of keyword followed by followed by a series of the same keywords followed by sequence numbers. Hence we have the situation keyword followed by possibility 14.
Input:

CODE(i) The name of the keyword.

TABLE(i) -IPOSMAIN

CODEMN(IPOSMAIN) Subkeyword that is used before the sequence number, in the example `P`.

TABMIN(i) Maximum number of integers that are allowed.

TABMAX(i) Maximum absolute value of the integers that are read.

Output

TABLE(i) If the keyword has been found then TABLE(i) gets the value IPOSUT (> 0), see TABOUT.
If the keyword is not found, TABLE(i) is left unchanged.

TABMAX(i) m, number of numbers read.

TABOUT(IPOSUT) to TABOUT(IPOSUT+m-1) contain the numbers read.

19 The input expected is of the shape `keyword = ( x1, y1, z1 )`
Special situation of keyword followed by a series of numbers possibly enclosed by brackets.
Input:

CODE(i) The name of the keyword.

TABLE(i) Some non-positive value.

TABMIN(i) Maximum number of numbers that are allowed.

TABMAX(i) Maximum absolute value of the numbers that are read.

Output

TABLE(i) If the keyword has been found then TABLE(i) gets the value IPOSUT (> 0), see TABOUT.
If the keyword is not found, TABLE(i) is left unchanged.

TABMAX(i) m, number of numbers read.

TABOUT(IPOSUT) to TABOUT(IPOSUT+m-1) contain the numbers read.

20 The input expected is of the shape `keyword` or `keyword keyword = i1, i2, ...`
Special situation of keyword or the same keyword followed by another special keyword followed by integers.
Input:

CODE(i) The name of the keyword.

TABLE(i) -IPOSMAIN

CODEMN(IPOSMAIN) Subkeyword that is used before the sequence number, in the example `P`.

TABMIN(i) Maximum number of integers that are allowed.

TABMAX(i) Maximum absolute value of the integers that are read.

Output

TABLE(i) If the keyword has been found followed by the second keyword, then TABLE(i) gets the value IPOSUT (> 0), see TABOUT.
If the keyword has been found but not the second keyword TABLE(i) gets the value -100000.
If the keyword is not found, TABLE(i) is left unchanged.

TABMAX(i) m, number of numbers read.

TABOUT(IPOSUT) to TABOUT(IPOSUT+m-1) contain the numbers read.

21 The input expected is of the shape `keyword (ax`by+cz=d)+
Special situation of keyword followed by (ax+by+cz=d), with a, b, c and d numbers. If a, b or c is omitted 1 or -1 is supposed.
Input:

CODE(i) The name of the keyword.

TABLE(i) Some non-positive value.

Output

TABLE(i) If the keyword has been found then TABLE(i) gets the value IPOSUT (> 0), see TABOUT.
If the keyword is not found, TABLE(i) is left unchanged.

TABOUT(IPOSUT) to TABOUT(IPOSUT+3) contain the values a, b, c and d in that sequence.

22 The input expected is of the shape `keyword [keyword1] [keyword2] = ( x1, y1, z1 )`
Special situation of keyword followed by possibly some other special keywords followed by a series of numbers.
Input:

CODE(i) The name of the keyword.

TABLE(i) -IPOSMAIN

CODEMN(IPOSMAIN) to CODEMN(IPOSMAIN+1) Subkeywords keyword1 and keyword2.

TABMIN(i) Maximum number of numbers that are allowed.

TABMAX(i) Maximum absolute value of the numbers that are read.

Output

TABLE(i) If the keyword has been found then TABLE(i) gets the value IPOSUT (> 0), see TABOUT.
If the keyword is not found, TABLE(i) is left unchanged.

TABMAX(i) m, number of numbers read.

TABOUT(IPOSUT) to TABOUT(IPOSUT+m-1) contain the numbers read.

23 The input expected is of the shape `keyword keyword`.
Special situation of keyword followed by by a special keyword that must be skipped.
Input:

CODE(i) The name of the keyword.

TABLE(i) -IPOSMAIN

CODEMN(IPOSMAIN) Subkeyword.

Output

TABLE(i) If the keyword has been found then TABLE(i) gets the value 1.
If the keyword is not found, TABLE(i) is left unchanged.

24 The input expected is of the shape `keyword [keyword1] [keyword2] = ( i1, i2, i3 )`.
This is exactly the same case as ITYPE(i)=22, however, in this case the numbers must be positive integers.
The input and output is exactly the same as for ITYPE(i)=22.

25 The input expected is of one of the following shapes:

```
keyword iseq1 = real
keyword iseq1 (value = real)
keyword iseq1 (func = i)
keyword iseq1 (time_func = i)
```

Special situation of keyword followed by sequence number followed by either a value or the possibilities value = v, func = ifunc, or time_func = ifunc. (See also ITYPE(i)=10 and ITYPE(i)=16).
Input:

CODE(i to $i + max - 1$) The name of the keyword, max times repeated.

TABLE(i to $i + max - 1$) Values that can not occur at output.

TABMIN(i) Minimum value of the sequence numbers.

TABMAX(i) Maximum value of the sequence numbers. Hence max = TABMAX(i)-TABMIN(i)+1.

Output

TABLE(i+k-TABMIN(i))(k = TABMIN(i), ... , TABMAX(i)) contains:
the value IFUNC (> 0), or IPOSUT (< 0), see TABOUT.
If a real value is found then this value is stored in TABOUT(-IPOSUT),
if `func = j` is found TABLE get the value `j`.
If `time_func = j` is found TABLE gets the value -100-`j`.
If the keyword is not found, TABLE is left unchanged.

26 The input expected is of one of the following shapes:

```
keyword = complex
keyword (value = complex)
keyword (func = i)
keyword (time_func = i)
```

with `complex` = real or (real,real).
Special situation of keyword followed by either a value or the possibilities value = v, func = ifunc, or time_func = ifunc, with v complex.
Input:

CODE(i) The name of the keyword.

TABLE(i) Some value that can not occur as output.

Output

TABLE(i) If the keyword has been found then TABLE(i) gets the value IFUNC (> 0), or IPOSUT (< 0), see TABOUT. If a complex value is found then this value is stored in TABOUT(-IPOSUT) and TABOUT(-IPOSUT+1). The first entry contains the real part, the second one the imaginary part. if `func = j` is found TABLE(i) get the value `j`.
If `time_func = j` is found TABLE(i) get the value -100-`j`.
If the keyword is not found, TABLE(i) is left unchanged.

TABOUT(-IPOSUT) to TABOUT(-IPOSUT+1 get the complex value read.

27 The input expected is of the shape keyword = series of items which may be of the following shapes:

```
real
keyword (value = real)
keyword (func = i)
keyword (time_func = i)
```

Input:

CODE(i) The name of the keyword.

TABLE(i) Some value that can not occur as output.

TABMIN(i) Number of values expected.

Output

TABLE(i) If the keyword has been found then TABLE(i) gets the value IPOSUT (> 0), see TABOUT.
If the keyword is not found, TABLE(i) is left unchanged.

TABOUT(IPOSUT+2×j-2) (j=1, 2, ..., TABIMIN(i)) gets the following value:
If value is found then 0 and TABOUT(IPOSUT+2×j-1) gets the value read.
If `func=j` is found the value j is stored, and
if `time_func=j` is found, the value -j-100 is stored.

28 The input expected is of the shape `keyword = i1, i2, i3, ...`, all integer
Special situation of keyword followed by series of integer numbers. See ITYPE(i)=11. The only difference is that the integers do not have to be positive.

29 The input expected is of the shape `keyword iseq = [keyword1] i1, i2, i3, ...`, all integer
Special situation of keyword followed by sequence number possibly followed by another keyword followed by integers.
Input:

CODE(i) The name of the keyword.

TABLE(i) -IPOSMAIN.

CODEMN(IPOSMAIN) Subkeyword.

TABMIN(i) Maximum number of integers that are allowed.

TABMAX(i) Maximum absolute value of the integers that are read.

Output

TABLE(i) If the keyword has been found then TABLE(i) gets the value IPOSUT (> 0), see TABOUT.
If the keyword is not found, TABLE(i) is left unchanged.

TABMAX(i) m, the number of integers that have been read.

TABOUT(IPOSUT) to TABOUT(IPOSUT+m-1 contain the values of the integers that have been read.

30 The input expected is of the shape `keyword iseq1 = c1, keyword iseq2 = c2, ...` Special situation of keyword followed by sequence number followed by keyword and integer number, for example obs i = cj or alternatively keyword followed by keyword and integer number for example obs = cj. i = 1 in that case Input:

CODE(i to $i + max - 1$) The name of the keyword, max times repeated.

TABLE(i to $i + max - 1$) Some values that are outside the range of the numbers `j1, j2, ....`

TABMIN(i) Minimum value of the sequence numbers.

TABMAX(i) Maximum value of the sequence numbers. Hence max = TABMAX(i)-TABMIN(i)+1.

TABMIN($i + 1$) Minimum value of the numbers `j1, j2, ....`

TABMAX($i + 1$) Maximum value of the numbers `j1, j2, ....`

ITYPE($i + 1$) IPOSMAIN, position in array CODEMN.

CODEMN(IPOSMAIN), special sub keyword.

Output

TABLE(i to $i + max - 1$) contain the values of the numbers `j1, j2, ..` at the positions corresponding to the sequence numbers. If a sequence number is not found in the input file, the input value of TABLE(i+iseq-1) is kept.

19.4.5 Subroutine INTX04

Description

Subroutine INTX04 is used to read a series of numbers.
The series is closed by a) sign, end of record or a text.
The reading of numbers is stopped if:

- a) sign is found.
- an end of record is found.
- maxnum numbers have been read.

After reading the numbers, ICOLUM in common block CSEPRD (19.4.1), points to the next position to be read.

Heading

```
subroutine intx04 ( rnumber, maxnum, inum )
```

Parameters

integer maxnum, inum

double precision rnumber(maxnum)

rnumber Array to be filled with the numbers that are read.

MAXNUM Maximum number of numbers that may be read.

INUM Actual number of numbers that are read.

Input

Parameter MAXNUM must have a value.

Output

Array RNUMBER has been filled.
INUM has got a value.

19.5 Special user subroutines

19.5.1 Subroutine USEROUT

Description

Subroutine USEROUT is a user written subroutine that must be provided if the user uses program SEPCOMP in combination with the input block STRUCTURE.
Each time in structure the command:

```
user_output, sequence_number = i
```

is given, subroutine USEROUT is called.
If the option `extra_scalars` is used, then USEROUTS is called instead of USEROUT.

Heading

```
CALL USEROUT ( KMESH, KPROB, ISOL, ISEQUENCE, NUMVEC )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), ISOL(5,∗), ISEQUENCE(∗), NUMVEC

KMESH Standard SEPRAN array containing information of the mesh. KMESH has been filled by the SEPRAN starting subroutine.

KPROB Standard SEPRAN array containing information of the problem definition. KPROB has been filled by the SEPRAN starting subroutine.

ISOL Array of size $5 \times MAXVEC$, where MAXVEC is defined in common block CUSCONS (User Manual, Section 1.6). ISOL contains references to the first NUMVEC solution arrays. With solution array we mean any array referenced in the input block STRUCTURE by Vj. So that may be a solution array, but also an array of special structure or an array that is created by manipulation of other "solution-type" arrays.
ISOL(.,j) corresponds to vector Vj.
If a specific solution array must be used in a SEPRAN subroutine, it may be addressed as
ISOL(1,j)
for example in the print subroutine PRINVC:

```
call princ ( inppri, kmesh, kprob, isol(1,j), icurvs, 'text' )
```

The user may perform any SEPRAN manipulation he wants with these solution arrays, however, he himself is responsible for the effects.

ISEQUENCE This array contains at least two positions.
Position 1 is equal to the parameter i after sequence_number in the input file. It may be used to distinguish between several calls.
Position 2 contains the number of extra integers as defined by `extra_integers = (i_1, i_2, ... )`.
If this number is larger than 0, the extra integers are stored in positions 3, 4, ...

NUMVEC Contains the highest sequence number of the solution vectors that are filled. This does not necessarily mean that all vectors V_1 to V_{NUMVEC} have been filled since it is possible that some sequence numbers have been skipped. Arrays that are not filled may be recognized by the fact that position (1,j) is equal to 0.

Input

The arrays KMESH, KPROB, ISOL and ISEQUENCE have been filled by program SEPCOMP.
NUMVEC has got a value by SEPCOMP.

Output

The user may change the contents of array ISOL, but not of the other arrays. As a consequence he may also change NUMVEC accordingly, provided NUMVEC does not exceed MAXVEC.

Subroutine layout

Subroutine USEROUT must be programmed as follows:

```
      subroutine userout ( kmesh, kprob, isol, isequence, numvec )
      implicit none
      integer kmesh(*), kprob(*), numvec, isol(5,*),
     +        isequence

      include 'SPcommon/comcons1'
      include 'SPcommon/cuscons'

         statements to manipulate isol and possibly numvec using standard
         SEPRAN subroutines as described in the Programmers Guide

      end
```

With respect to common block CUSCONS the user is referred to the Users Manual Section 1.6.

Remark

The user may utilize USEROUT not only with respect to the solution arrays, he may also change the values of the scalars in this subroutine.

19.5.2 Subroutine USEROUTS

Description

Subroutine USEROUTS is exactly equal to subroutine USEROUT, except for the extra parameter SCALS.
USEROUTS is called if the option `extra_scalars` is used in the USER OUTPUT option.

Heading

```
CALL USEROUTS ( KMESH, KPROB, ISOL, ISEQUENCE, SCALS, NUMVEC )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), ISOL(5,∗), ISEQUENCE(∗), NUMVEC

DOUBLE PRECISION SCALS(∗)

KMESH, KPROB, ISOL, ISEQUENCE, NUMVEC See USEROUT

SCALS Array containing the values of the extra scalars defined by the user in the sequence given by the user. Changing these values in USEROUTS in array SCALS does not make sense, since SCALS is not copied back into array SCALARS.

Input

The arrays KMESH, KPROB, ISOL, SCALS and ISEQUENCE have been filled by program SEPCOMP.
NUMVEC has got a value by SEPCOMP.

Output

The user may change the contents of array ISOL, but not of the other arrays. As a consequence he may also change NUMVEC accordingly, provided NUMVEC does not exceed MAXVEC.

Subroutine layout

Subroutine USEROUTS must be programmed as follows:

```
      subroutine userouts ( kmesh, kprob, isol, isequence, scals,
     +                      numvec )
      implicit none
      integer kmesh(*), kprob(*), numvec, isol(5,*), isequence
      double precision scals(*)

      include 'SPcommon/comcons1'
      include 'SPcommon/cuscons'

         statements to manipulate isol and possibly numvec using standard
         SEPRAN subroutines as described in the Programmers Guide

      end
```

20 Writing results to files and reading from files

20.1 Introduction

In this chapter we will describe the various possibilities to write SEPRAN arrays to a file or to read them from a file. Several of these possibilities are already described in the users manual and in most programs it is sufficient to use the options described there. However, in this chapter we give a complete survey of all the possibilities and also describe the subroutines that perform the reading, writing, opening and closing tasks. Moreover the structure of the SEPRAN files will be described.

SEPRAN contains two different ways of writing results to a file, except of course the writing of the vectors to standard output by a print subroutine.
The first way is to write solution arrays to the file sepcomp.out. This option is especially meant for usage in combination with the SEPRAN post processing program SEPPOST, but may in fact be used for any application. This option is strictly serial of nature and can only be used to write the solution of one single run.
The next way is to write arrays to a direct access file which allows for random access of arrays. This option is meant for saving of solutions and so on for later use, for example to restart a computation or to create an initial condition. Of course this option may also be used for postprocessing purposes.
The third way is to create so-called AVS output files. These files are written in so-called AVS format and make only sense if you have the postprocessing program AVS at your disposal.

Besides that SEPRAN offers the user the possibility to read his own data into a SEPRAN vector, provided he gives this data in the same sequence as the nodal points. This possibility is meant for the case that the user has given all nodal points and SEPRAN has made a mesh from these points using the option `nodal_points` as described in the Users Manual Section 2.7.

Let us consider the possibilities into more detail.

Writing and reading from the file sepcomp.out

In fact in a standard run of sepcomp in general the results are written to the file sepcomp.out. This writing is completely organized by the keyword OUTPUT in the input file for sepcomp. If this keyword is not present only the standard solution is written.
The part OUTPUT is activated by an explicit command in the part STRUCTURE or the command output = i in the part NONLINEAR_EQUATIONS. If no explicit call is present, OUTPUT is activated at the end of the sepcomp run.
The program SEPPOST automatically reads the files sepcomp.inf and sepcomp.out completely and uses the results for the postprocessing.
In a user written program it is also possible to write solutions to the file sepcomp.out and sepcomp.inf or to read information from these files. For that purposes the subroutines OUTPUT, OUTTIM and at a lower level PRINSL and READSL have been developed. Opening and closing of these files is treated in Section 19.2.

Writing and reading from the direct access file sepranback

A complete other way of storing results is to write specific arrays to the SEPRAN direct access file. This option may also be activated by the input of program SEPCOMP using the option SAVE in the various input blocks. Reading of these arrays is activated by the option READ. In this way results of computations can be used for restart purposes, to create boundary or initial condition for a new computation or just for postprocessing.
The SEPRAN direct access file contains a complete administration which makes it easier for the user to find specific arrays. To open or close the direct access file in a user written program consult Section 19.2.
The SEPRAN direct access file is available in 2 versions which are mutually exclusive. In the present version the administration is much more sophisticated than in the previous version. For that reason the old and new version can not be used together except when the user takes his own

precautions. Writing to the new version is done by the subroutine WRITSB and reading from this file by READSB. For the old version the subroutines WRITBS and READBS are available.

This chapter is subdivided into a number of parts, each of which have been divided in several subsections. The following sections are available.

20.2 sepcomp.out describes the serial file sepcomp.out and also the subroutines that may be used to read from this file and to write to this file.

20.3 sepranback describes the direct access file sepranback and also all subroutines related to this file.

20.4 AVS describes the subroutines that create AVS input files. Mark that you can only use these files if you have the postprocessings package AVS at your disposal. AVS is not a part of SEPRAN nor is delivered with SEPRAN.

20.5 Reading own data describes a subroutine that may be used to read user data into a vector provided these data are given in the sequence of the nodal points.

20.2 Writing to the file SEPCOMP.OUT

In this Section we treat the subroutines that can be used explicitly to write the results computed in the computational program to the file and sepcomp.out. This file is used by program SEPPOST for post-processing purposes.
The following subroutines are available:

OUTPUT (20.2.1) Standard subroutine to write solutions to the files sepcomp.xxx. Information is read from the standard input file.

OUTTIM (20.2.2) Extension of OUTPUT especially meant for time-dependent problems. The input is only read at the users wish and is kept for the next call.

OUTPOS (20.2.3) is exactly the same as OUTTIM, but besides that has two extra parameters IUSER and USER, which may be used in the computation of derived quantities.

OUTSOL (20.2.4) Is the most general one of the series of subroutines. It does not only perform the same tasks as OUTPOS, moreover it allows the reading of the input to be performed by the SEPRAN starting subroutine, or alternatively allows input through the parameter list without any reading at all.

In Subsection 20.2.5 the contents of the file sepcomp.out is given.

20.2.1 Subroutine OUTPUT

Description

The purpose of subroutine OUTPUT is two-fold. In first instance OUTPUT is meant to write the solution to a file named SEPCOMP.OUT. On the other hand derived quantities as for example the pressure, the gradient of the solution, the stream function etcetera are computed and written to the same file SEPCOMP.OUT.

Heading

```
subroutine output ( kmesh, kprob, isol )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), ISOL(5,∗)

KMESH Standard SEPRAN array in which information of the mesh is stored.

KPROB Standard SEPRAN array containing information concerning the problem definition.
KPROB should have been filled by a SEPRAN starting subroutine or subroutine PROBDF (4.4).

ISOL Array containing information of the solution vector. Output of a solution subroutine.
It may be possible that ISOL contains more than one solution, in which case ISOL is a more dimensional array. Otherwise ISOL has length 5.

Input

The arrays KMESH, KPROB and ISOL must have been filled.
Furthermore OUTPUT requires input from the standard input file as described in the Users Manual Section 3.2.13.

Output

The solution and, if necessary, derived quantities are written to the file named sepcomp.out.

20.2.2 Subroutine OUTTIM

Description

The purpose of subroutine OUTTIM is two-fold. In first instance OUTTIM is meant to write the solution at a certain time to a file named SEPCOMP.OUT. On the other hand derived quantities as for example the pressure, the gradient of the solution, the stream function etcetera are computed and written to the same file SEPCOMP.OUT.
The essential difference with subroutine OUTPUT (20.2.1) is that OUTTIM may be called more than once and that it does not close the SEPRAN run.

Heading

```
subroutine outtim ( time, kmesh, kprob, isol )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), ISOL(5,∗)

DOUBLE PRECISION TIME

KMESH, KPROB, ISOL See subroutine OUTPUT (20.2.1).

TIME The actual time corresponding to the solution to be written. This time is stored in the file SEPCOMP.OUT and is used for post-processing purposes.

Input

The arrays KMESH, KPROB and ISOL must have been filled.
TIME must have a value.
Furthermore the first time OUTTIM is called, it requires input from the standard input file as described in the Users Manual Section 3.2.13.
This input is reused in all succeeding calls of OUTTIM.

Output

The solution and, if necessary, derived quantities are written to the file named sepcomp.out.

20.2.3 Subroutine OUTPOS

Description

OUTPOS is completely identical to subroutine OUTTIM (20.2.2), with the exception of the two extra parameters IUSER and USER.
These parameters may be used for the computation of derived quantities as described for OUTPUT (20.2.1).

Heading

```
subroutine outpos ( time, kmesh, kprob, isol, iuser, user )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), ISOL(5,∗), IUSER(∗)

DOUBLE PRECISION TIME, USER(∗)

TIME, KMESH, KPROB, ISOL See subroutine OUTTIM (20.2.2).

IUSER, USER User arrays that may used to compute derived quantities.
Actually, depending on the input as stored in the input file, OUTPUT and its related subroutines call subroutine DERIV to compute derived quantities. The parameters IUSER and USER are passed undisturbed to DERIV.

Input

The arrays KMESH, KPROB and ISOL must have been filled.
TIME must have a value.
If necessary IUSER and USER must have been filled
Furthermore the first time OUTPOS is called, it requires input from the standard input file as described in the Users Manual Section 3.2.13.
This input is reused in all succeeding calls of OUTPOS.

Output

The solution and, if necessary, derived quantities are written to the file named sepcomp.out.

20.2.4 Subroutine OUTSOL

Description

OUTSOL is identical to subroutine OUTPOS (20.2.3), with the exception of the extra parameter IINOUT.
This parameter is used to decide whether new input must be read from the standard input file, or old input is reused.
Also it is possible to read the input for OUTSOL by the SEPRAN starting subroutines.

Heading

```
subroutine outsol ( kmesh, kprob, isol, iinout, time, iuser, user )
```

Parameters

INTEGER KMESH(∗), KPROB(∗), ISOL(5,∗), IUSER(∗), IINOUT(∗)

DOUBLE PRECISION TIME, USER(∗)

TIME, KMESH, KPROB, ISOL, IUSER, USER See subroutine OUTPOS (20.2.3).

IINOUT Input array defined by the user. With this array the user may indicate which actions he wants to be performed by this subroutine.
IINOUT must be filled as follows:

1 Last entry number that has been filled by the user. If 0 or 1 only defaults are used.

2 IREAD Defines how the input must be read.
Possible values:

- 0 All SEPRAN input has been read by subroutine SEPSTN (4.2.3) until END_OF_SEPRAN_INPUT or end of file has been found.
  This input is used
- 1 The input is read as described for OUTPUT.
- 2 The last input read as described for OUTPUT. is used (equivalent with OUTTIM).

Default value: 0

3 ISEQNR, This parameter is only used if IREAD=0. It indicates the sequence number of the input for essential boundary conditions to be used.
Default value: 1

4 USE_IUSER, indicates if IUSER and USER from the parameter list must be used (1) or not (0).
Default value: 0

5 ITER, Iteration sequence number.
Default value: 1 (i.e. first iteration).

Input

The arrays IINOUT, KMESH, KPROB and ISOL must have been filled.
TIME must have a value.
If necessary IUSER and USER must have been filled
Depending on array IINOUT OUTSOL requires input from the standard input file as described in the Users Manual Section 3.2.13.

Output

The solution and, if necessary, derived quantities are written to the file named sepcomp.out.

20.2.5 The file sepcomp.out

The file `sepcomp.out` is a sequential file, which may be either formatted or unformatted. The file is organized in records. All unformatted text records have a length of 80 characters.

Contents

Record 1 Text record with the text:

```
//Version:  version_number, isubr, leniinput
```

Record 2 ivers, isubr, leniinput
where ivers is the version number, isubr a code indicating by which subroutine sepcomp.out is made and leniinput the length of the array iinput, stored in record 4.

Record 3 Text record with the text:

```
//Iinput
```

Record 4 Array iinput of length leniinput. This array is used as input for subroutine PROBDF. In the formatted case it may actually consist of more records.

Record 5 Text record with the text:

```
//General_info:   numvec, lenipost, lenrpost
```

Record 6 numvec, lenipost, lenrpost
where numvec is the number of different type of vectors that may be written into the file `sepcomp.out`.
lenipost defines the length of the array ipost. If lenipost = 0, there is no array ipost.
lenrpost defines the length of the array rpost. If lenrpost = 0, there is no array rpost. is the version number, isubr a code indicating by which subroutine is made and leniinput the length of the array iinput, stored in record 4.

Record 7 Text record with the text:

```
//Names_of_vectors
```

This record is only present if numvec>0.

Next numvec records names(i), with names(i) the name of the i^{th} vector. names(i) has length 32 characters. (i from 1 to numvec)

Next records For each vector that is stored in sepcomp.out the following information is stored:

First record Text record with the text:

```
//Vector_name: xxxxxxxxx
```

`xxxxxxxxx` is the name of the vector. It is one of the names stored in array names.

Second record Text record with the text:

```
//yyyyyyyy
```

`yyyyyyyy` may be either `Points`, `Elements` or `Faces`.
`Points` means that the vector is defined in the nodes of the mesh.
`Elements` means that the vector is defined per element of the mesh.
`Faces` means that the vector is defined per face (3D) or edge (2D) of the mesh.

Next 4 records Text records with the text:

```
//Comment: the next 3 data lines contain
//Comment: sequence_number, iteration_number   , time_level, 7*0
//Comment: isol (5 integers for seppost)
//Comment: nlines, ncomponents, nglobal
```

Next record iseq, iter, t, 0, 0, 0, 0, 0, 0
with iseq a sequence number referring to names(iseq+1)
iter an iteration number
t the actual time.

Next record Array or isol or ivectr as described in Sections 24.4 and 24.8

Next record nlines, ncomponents, nglobal
nlines denotes the number of lines (Records) written with actual data and ncomponents the number of reals (or complex values) per vector.
nglobal denotes the number of global unknowns.

Next record Text record with the text:

```
//No   Component_1    Component_2, ....
```

Here `Component_1` denotes the name of the first component, `Component_2` of the second one and so on.

In the unformatted case this record consist of a string of 8 characters, followed by a string of 16 $\times$ `ncomponents` characters.

Next nlines records Records with actually data with respect to the vector. Per record this consists of one integer and ncomponents double precision reals or complex numbers.
The integer denotes a sequence number, which refers to the node, element or face.
If a component has no value in a point, the value is set equal to `-rinfin` (see common block `cmcdpr`, Section 21.4).
Usually this is the value `-1d78`.

Next record If nglobal > 0, there is an extra record containing the integer nlines+1, followed by nlgobal double precision reals or complex numbers.

20.3 Writing and reading from the file SEPRANBACK

In this Section we describe the subroutines that may be used to write or read from the backing storage file SEPRANBACK. At this moment only the old subroutines are described, the new ones are under development.
It concerns the following subroutines:

Subroutine WRITBS (20.3.1) Write array to file sepranback.

Subroutine READBS (20.3.2) Read array from file sepranback.

Subroutine WRITB1 (20.3.3) Update the administration of sepranback.

20.3.1 Subroutine WRITBS

Description

Subroutine WRITBS writes arrays on a permanent file on backing storage (file 2). These arrays may be read by subroutine READBS (20.3.2). When subroutine WRITBS is used, the backing storage file sepranback must have been opened. This may be done in the following ways:

- By using the input block START as described in Section 3.2.1 of the Users Manual if a standard SEPRAN SEPSTx starting subroutine is called.
  In the block START, the input DATABASE = d, with d equal to *old* or *new* must be given.
- If subroutine START (Section 4.3) is used in stead of the SEPSTx subroutine, the parameter JSEPBC must have the value 1 or -1.
- An other possibility is explicitly calling the opening subroutine OPENF2 (Section 19.3.2)

In order to save the results, subroutine FINISH (19.3.1) must be called with ISTOP = 1.

Call

```
CALL WRITBS ( ICHOICE, INUM, NUMARR, NAME, IARRAY, IHELP )
```

Parameters

INTEGER ICHOICE, INUM, NUMARR, IARRAY(∗), IHELP(∗)

CHARACTER (∗) 10 NAME

ICHOICE Choice parameter. This parameter indicates whether the array is a SEPRAN array (ICHOICE = 0) or not (ICHOICE ≠ 0).
When IARRAY corresponds to a user array, the value of —ICHOICE— denotes the length of this array, and the sign of ICHOICE denotes whether it is a real array (ICHOICE<0) or an integer array (ICHOICE>0).

INUM Choice parameter. Possibilities:
When INUM ≤ 0 the array is written on the next free positions on backing storage. At most 500 arrays can be written on the permanent file.
When INUM = -1 the subroutine prints a text of the following form:
Array "NAME" is stored as array number "NUMARR"
where "NAME" denotes the value of NAME and "NUMARR" denotes the value of NUMARR.
When INUM > 0 the array is written on positions of a preceding written array, with array number INUM. See also NUMARR. The length of the new and old array may be different. This possibility may also be used when there is no preceding array with this number.

NUMARR Each array that is written by WRITBS gets a sequence number, the first array gets number 1, the second number 2 etc. At output NUMARR gets the value of this sequence number. Therefore NUMARR must be a variable and not a constant.

NAME Name of the array to be written to backing storage. Name is a character constant of at most 10 characters. NAME is only used for printing purposes, when INUM = -1.

IARRAY Array to be written on backing storage.
IARRAY may be both a real or an integer array.
SEPRAN arrays are always integer arrays.

IHELP For some SEPRAN arrays an extra array is needed. This extra array, given below for the various SEPRAN arrays, must be used instead of IHELP.
When no extra array is needed, a dummy parameter can be used.
Extra arrays necessary for SEPRAN arrays:

IARRAY is KMESH:	IHELP is dummy
IARRAY is KPROB:	IHELP is KMESH
IARRAY is ISOL:	IHELP is dummy
IARRAY is INTMAT:	IHELP is KPROB
IARRAY is MATR:	IHELP is dummy

Input

ICHOICE and INUM must have a value.
When INUM = -1, NAME must have a value.
Array IARRAY must have been filled.

Output

IARRAY and when IARRAY is a SEPRAN array its corresponding part in the buffer space, is written on file 2.
NUMARR has got a value.

20.3.2 Subroutine READBS

Description

This subroutine reads arrays that have been written by WRITBS (20.3.1), in the same program or a preceding program, from the permanent file (file 2) on backing storage.
When subroutine READBS is used, the file sepranback must have been opened by one of the methods given in WRITBS (20.3.1).

Call

```
CALL READBS ( NUMARR, IARRAY, IHELP )
```

Parameters

INTEGER NUMARR, IARRAY(∗), IHELP(∗)

NUMARR Sequence number, created by WRITBS. See subroutine WRITBS (20.3.1).

IARRAY IARRAY contains at output the array to be read from file 2. IARRAY may be a real or an integer array. It must have the same type as the array corresponding to NUMARR that has been written by WRITBS (20.3.1).

IHELP Help array. IHELP has exactly the same meaning as in subroutine WRITBS (20.3.1).

Input

NUMARR must have a value.
Depending on the type of array IARRAY, array IHELP must have been filled.

Output

Array IARRAY has been filled.

20.3.3 Subroutine WRITB1

Description

Subroutine WRITB1 writes the organisation of file 2, as stored in common block CFILE2 (21.9), to the first record of file 2, thus saving all information that has been written to file 2.

Call

```
CALL WRITB1
```

Output

Common block CFILE2 (Section 21.9) has been written to the first record of file 2.

20.4 Creating AVS input files (subroutines PRIAVS and PRTAVS)

In this section subroutines to create AVS input files are discussed. The SEPRAN solution array as well as information of the mesh is written to a set of files sepavs.xxx or sepavs_xxx.yyy. These files may be read by the postprocessing program AVS.
The use of these subroutines makes only sense if you have the AVS package at your disposal.
AVS is not a part of SEPRAN and is not delivered by SEPRA.

At this moment there are two subroutines that may be used to create AVS files:

PRIAVS (20.4.1) Standard subroutine to create AVS files. It is able to handle both two and three-dimensional meshes. Internally all elements are subdivided into either linear triangles of linear tetrahedrons.

PRTAVS (20.4.2) Special subroutine to translate (x, y, t) data towards a 3D mesh in AVS. The mesh consists of prisms.
Using the AVS slicer one can see how a quantity changes in time.

20.4.1 subroutine PRIAVS

Description

Standard subroutine to create AVS files. It is able to handle both two and three-dimensional meshes. Internally all elements are subdivided into either linear triangles of linear tetrahedrons.

Call

```
CALL PRIAVS ( KMESH, KPROB, ISOL, ITIME, ISEQ, TIME )
```

INTEGER KMESH(∗), KPROB(∗), ISOL(5), ISEQ

DOUBLE PRECISION TIME

KMESH Standard SEPRAN array, containing information of the mesh.

KPROB Standard SEPRAN array, containing information of the problem definition.

ISOL Standard SEPRAN array containing information of the solution array or vector of special structure to be written.

ITIME Integer time sequence number. This number is used to create the suffix yyy in the file name.

ISEQ Sequence number of the vector to be written. This number is used to create the suffix xxx in the file name.

TIME Actual time

Input

The arrays KMESH, KPROB and ISOL must have been filled.
The parameters ISEQ, ITIME and TIME must have a value.

Output

A file sepavs_xxx.yyy has been created.

20.4.2 subroutine PRTAVS

Description

Special subroutine to translate (x, y, t) data towards a 3D mesh in AVS. The mesh consists of prisms.
Using the AVS slicer one can see how a quantity changes in time.
This subroutine writes a solution array or array of special structure to the file sepavs.xxx.
This possibility is meant for a SEPRAN-AVS interface.
The 2D SEPRAN mesh is mapped onto a mesh of linear triangles. The 2D mesh is repeated ntime times to construct a 3D mesh consisting of prisms.

Call

```
CALL PRTAVS ( ICHPRT, KMESH, KPROB, ISOL, NTIME, ISEQ, TIME, SCALE )
```

INTEGER ICHPRT, KMESH(∗), KPROB(∗), NTIME, ISOL(5,NTIME), ISEQ

DOUBLE PRECISION TIME(NTIME), SCALE

ICHPRT choice parameter with the following options:

- 0 Only time axis is scaled. Scaling according to the minimum of xmax-xmin, ymax-ymin.
- 1 time axis and either x or y is scaled. The coordinate direction with minimum width and the time axis is scaled to SCALE with maximum width.

KMESH Standard SEPRAN array, containing information of the mesh.

KPROB Standard SEPRAN array, containing information of the problem definition.

ISOL Standard SEPRAN array containing information of the various solution arrays or vectors of special structure to be written.
Each ISOL(.,k) refers to a new time step.

ISEQ Sequence number of the vector to be written. This number is used to create the suffix xxx in the file name.

NTIME Number of time steps stored in ISOL and TIME.

TIME Array of actual times

SCALE scale factor for ichprt=1

Input

The arrays TIME, KMESH, KPROB and ISOL must have been filled.
The parameters ISEQ, NTIME and SCALE must have a value.

Output

A file sepavs.xxx has been created.

20.5 Reading a vector into SEPRAN (subroutine READVEC)

Description

Subroutine READVEC may be used to read user data into a SEPRAN vector. This vector is created as a vector of special structure defined per point (type 115). It is necessary that the users provides the data in exactly the sequence of the nodal points. Hence the user must know this sequence. In general this is only possible in case of a structured mesh created by for example RECTANGLE or BRICK, or if the user has given all nodal points of the mesh himself. This is for example possible if the user has created the mesh by the option `nodal_points` as described in the Users Manual Section 2.7.

Call

```
CALL READVEC ( KMESH, KPROB, ISOL, NUMBER, FILENM )
```

INTEGER KMESH(∗), KPROB(∗), ISOL(5), NUMBER

CHARACTER ∗ **(**∗**)** FILENM

KMESH Standard SEPRAN array corresponding to the mesh.

KPROB Standard SEPRAN array corresponding to the problem definition.

ISOL Standard SEPRAN array containing information of the vector to be read. This array is created as a vector of special structure of type 115.

NUMBER defines the number of degrees of freedom per node for the vector to be read.
Mark that the problem description must contain vectors of special structure with NUMBER degrees of freedom per point.
Elements of type number 800 (corresponding to the default problem description) have vectors of special structure with 1 to 10 degrees of freedom per point. If you need more than 10 degrees of freedom per point, it is necessary to define extra vectors of special structure as described in the Users Manual, Section 3.2.2.

FILENM Name of the file from which the data must be read.

Input

The arrays KMESH and KPROB must have been filled.
The parameter NUMBER and FILENM must have a value.

Furthermore READVEC reads input from the file FILENM. This file must be an ASCII file, containing the vector to be read in the sequence:
degrees of freedom of node 1, degrees of freedom of node 2 and so on.
Hence first all degrees of freedom of the first node (in natural sequence) then of second node etc.
The nodes are read in the sequence of the mesh, hence the input must be given in the sequence of the mesh nodes.
The numbers are read according to standard FORTRAN free format, i.e. the number of numbers per line is arbitrary. No texts are allowed in this file, not even comments.

Output

The vector is read and the solution is stored in the array corresponding to ISOL.
Array ISOL has been filled.

21 The SEPRAN common blocks

21.1 Introduction

In this chapter we will describe a number of SEPRAN common blocks. In general a SEPRAN user does not have to know anything about the structure and contents of the SEPRAN commons, however, for a specific application it might be useful to know the exact structure.
Blank common is treated in Section 1.2. In this chapter we deal with the following commons blocks:

21.2 Common CBUFFR contains variables related to blank common.

21.3 Common CCONST contains variables related to error messages.

21.4 Common CMCDPR contains some important machine dependent real constants.

21.5 Common CMCDPI contains unit numbers to use for certain standard in- and output files.

21.6 Common CARRAY contains information of arrays that are stored in IBUFFR.

21.7 Common CINOUT contains parameters to indicate how much standard SEPRAN output is required.

21.8 Common CSTAND contains information concerning the file with reference number 1.

21.9 Common CFILE2 contains information with respect to the old version of the SEPRAN backing storage file.

21.10 Common CFILE3 contains information concerning the temporary file with reference number 3.

21.11 Common CPLOT contains some parameters for the plot subroutines.

21.12 Common CGRENS contains Information about the axis to be plotted.

21.13 Common CMCDPP contains machine-dependent plotting parameters.

21.14 Common CACTL contains element dependent information for the various element subroutines.

21.15 Common CPLAF is meant for communication between the low-level plotting routines.

21.16 Common CONSTA contains some general double precision constants.

21.17 Common CMACHN is used as common block for determination of computer system as well as some machine-dependent quantities and references.

21.18 Common CMACHT contains computer (machine) and terminal information in character and strings.

21.19 Common CSEPCM contains a series of texts that can be used in the SEPRAN programs.

21.20 Common CSEPINT contains a series of integers that can be used in the SEPRAN programs.

For all other commons the user is referred to the directory SPHOME/common, where SPHOME stands for SEPRAN home directory. In that directory you can find a complete description of all common blocks used in SEPRAN.

21.2 Common CBUFFR

Description

Common block CBUFFR contains several variables that are related to the arrays IBUFFR and BUFFER in blank common.

Declaration

```
integer nbuffr, kbuffr, intlen, ibfree
common /cbuffr/ nbuffr, kbuffr, intlen, ibfree
```

Parameters

nbuffr Declared length of array IBUFFR

kbuffr Last position used in IBUFFR

intlen Length of a real variable divided by the length of an integer variable. For most computers INTLEN has the value 2. However, for for example a CRAY INTLEN=1. INTLEN is set by the starting subroutine.

ibfree Next free position in array IBUFFR.

21.3 Common CCONST

Description

Common block CCONST contains only two parameters directly related to the error messages.

Declaration

```
integer ierror, nerror
common /cconst/ ierror, nerror
```

Parameters

ierror contains the error number of the last error occurred. If ierror=0, no error has been found.

nerror contains the number of errors found.

21.4 Common CMCDPR

Description

Common block CMCDPR contains some important machine dependent real constants.

Declaration

```
double precision epsmac, sqreps, rinfin
common /cmcdpr/ epsmac, sqreps, rinfin
```

Parameters

epsmac Machine accuracy, i.e. relative accuracy of double precisions. In case INTLEN=1 in common CBUFFR it is the relative accuracy of reals. For most computers epsmac = 10^{-15}. This parameter is made equal to the parameter *machine accuracy* in the sepran.env file (Users Manual Section 1.5).

sqreps Square root of machine accuracy or half precision. For most computers sqreps = 10^{-6}. This parameter is made equal to the parameter *accuracy of a half real* in the sepran.env file.

rinfin Approximation of infinity, i.e. the largest double precision (or real if INTLEN=1) permitted on this computer.
This parameter is made equal to the parameter *approximation of infinity* in the sepran.env file.

21.5 Common CMCDPI

Description

Common block CMCDPI contains unit numbers to use for certain standard in- and output files.

Declaration

```
integer irefwr, irefre, irefer
common /cmcdpi/ irefwr, irefre, irefer
```

Parameters

irefwr Unit number to use for "normal" writes
This parameter is made equal to the parameter *unit number for writing of SEPRAN input* in the sepran.env file (Users Manual Section 1.5).

irefre Same for standard reads (mostly keyboard input)
This parameter is made equal to the parameter *unit number for reading of SEPRAN input* in the sepran.env file.

irefer Same for error messages
This parameter is made equal to the parameter *unit number for error messages* in the sepran.env file.

21.6 Common CARRAY

Description

Common block CARRAY contains information of arrays that are stored in IBUFFR.

Declaration

```
integer iinfor, infor
common /carray/ iinfor, infor(3,1500)
```

Parameters

iinfor Number of arrays that are stored in IBUFFR. At most 1500 arrays can be stored, hence $0 \leq$ IINFOR ≤ 1500.

infor Information concerning the storage of arrays in IBUFFR is stored in the following way:

Pos. (1,i) If > 0: array with sequence number i is stored in IBUFFR from position infor(1,i). If < 0: array i is stored in file 3 from record number -infor(1,i)

Pos. (2,i) Length of the array in words (integers).

Pos. (3,i) Priority number. Is used to decide which arrays are put to file 3. $0 \leq$ prio ≤ 10. Arrays with priority 100 are always kept in-core

21.7 Common CINOUT

Description

Common block CINOUT contains parameters to indicate how much standard SEPRAN output is required.

Declaration

```
integer ifree, ioutp, itime
common /cinout/ ifree, ioutp, itime
```

Parameters

ifree Not used.

ioutp Indication of the amount of print output that must be performed by SEPRAN. Possible values:

-1 No SEPRAN output except for error messages and output explicitly required.

0 Minimum output

1 Information of dimensions etc. is printed

2 See 1, also all reads from and writes to backing storage are printed.

3 See 2, also all calls to a memory management subroutine are printed.

itime Indication if the CPU-time must be printed for each main subroutine (1) or not (0).

21.8 Common CSTAND

Description

Common block CSTAND contains information concerning the file with reference number 1. In this file, information concerning the standard elements is stored

Declaration

```
integer nrec1, irecp1, irecp2, iref1
common /cstand/ nrec1, irecp1, irecp2, iref1
```

Parameters

nrec1 Length of the records on file 1 in words.
This parameter is made equal to the parameter *Record length for file 1* in the sepran.env file (Users Manual Section 1.5).

irecp1 Record number of the last record that has been read as first part.

irecp2 Record number of the last record that has been read as second part.

iref1 Reference number for file 1.
This parameter is made equal to the parameter *Unit number for file 1* in the sepran.env file.

21.9 Common CFILE2

Description

Common block CFILE2 contains information concerning the old version of the sepran backing storage file with reference number 2. It is only used by the subroutines related to WRITBS and READBS.

Declaration

```
integer nrec2, ifree2, irec2, iref2, inew, iwork3
common /cfile2/ nrec2, ifree2, irec2, iref2, inew, iwork3(1000)
```

Parameters

nrec2 Length of the records on file 2 in words.
This parameter is made equal to the parameter *Record length SEPRAN backing storage file* in the sepran.env file (Users Manual Section 1.5).

ifree2 First free record on file 2.

irec2 The value of IREC2 is equal to the last record number used plus one.

inew Next free array number to be stored in array IWORK3.

iwork3 In this array of length 1000 information is stored of at most 500 arrays that have been written to file 2.
iwork3(2∗i-1) gives the record number from which the array is stored.
iwork3(2∗i) gives the length of the array in words.

21.10 Common CFILE3

Description

Common block CFILE3 contains information concerning file with reference number 3. This file is used as general backing storage file.
It is used as temporary file only.

Declaration

```
integer nrec3, ifree3, irec3, iref3
common /cfile3/ nrec3, ifree3, irec3, iref3
```

Parameters

nrec3 Length of the records on file 3 in words.
This parameter is made equal to the parameter *Record length for file 1* in the sepran.env file (Users Manual Section 1.5).

ifree3 First free record on file 3.

irec3 The value of irec3 is equal to the last record number used plus one.

iref3 Reference number for file 3.

21.11 Common CPLOT

Description

Common block CPLOT contains some parameters for the plot subroutines.

Declaration

```
      integer jmax, jmark, jframe, jtimes
      double precision xmin, xmax, ymin, ymax, zmin, zmax
      common /cplot/ xmin, xmax, ymin, ymax, zmin, zmax,
     +               jmax, jmark, jframe, jtimes
```

Parameters

xmin,xmax,ymin,ymax,zmin,zmax Minimum respectively maximum x, y and z-values of the region to be considered.
Hence only the points in the plotting area $xmin \leq x \leq xmax$, $ymin \leq y \leq ymax$, $zmin \leq z \leq zmax$ are plotted.

jmax Indication if the parameters xmin, xmax, ymin and ymax must be computed by the subroutine (0) or are given by the user (1).

jmark Flags, used to suppress drawing of scale-values. The following values are available:

0-99	Depends on the specific subroutine.
100	Suppress values along the x-axis.
1000	Same along left-hand y-axis.
10000	Same along right-hand y-axis (if selected).

jframe Parameter to indicate whether a frame work will be plotted as well as axis with scales. Possible values:

≥ 0	(standard) A plot frame is plotted without scales.
-1	No plot frame nor scales.
-2	A plot frame is plotted as well as axis with scales.
-3	No plot frame is plotted, the axis are plotted with scales.

The following texts are plotted in the frame-work depending on the value of JFRAME:

1	No text.
2	MESH.
3	FACTOR.
4	BOUNDARIES.
≥ 10	Submesh with sequence number JFRAME-10.

jtimes Parameter to indicate whether one or more plots are made in one picture. Possibilities:

0	Only one plot is made.
1	First plot in a sequence.
2	Next plot in a sequence.
3	Final plot in a sequence.
≥ 11	See jtimes-10, however, the plotting of axis and arrows in the standard plotting subroutines (except PLOTFN) is suspended. This possibility is meant for program SEPPOST. In this program the axis and arrows are plotted later on.

21.12 Common CGRENS

Description

Common block CGRENS contains information about the axis to be plotted.

Declaration

```
      integer ichoicex, ichoicey, isigx, isigy, nstepx, nstepy
      double precision axmin, axmax, aymin, aymax
      common /cgrens/ axmin, axmax, aymin, aymax, ichoicex, ichoicey,
     +                isigx, isigy, nstepx, nstepy
```

Parameters

axmin,axmax,aymin,aymax These parameters indicate the smallest respectively largest values to be plotted along the axis in x- and y-direction.

ichoicex Indication if axmin and axmax must be computed (-1) or not (Any other value). When > 0 then numbers will be plotted in fixed-point format, else a floating point format will be used.

ichoicey Same for the y-axis.

isigx Number of significant digits (floating) or decimal places (fixed point) along x-axis.

isigy Same for the y-axis.

nstepx Number of scale values to be plotted along x-axis.

nstepy Same for the y-axis.

21.13 Common CMCDPP

Description

Common block CMCDPP contains machine-dependent plotting parameters.

Declaration

```
integer irotat
double precision cm, small, wide, alengm
common /cmcdpp/  cm, small, wide, alengm, irotat
```

Parameters

cm Multiplication factor to transfer plotting units in centimeters.

small Width of small paper in centimeters.

wide Width of wide paper in centimeters.

alengm Maximal length of plot paper in centimeters

irotat Indication whether plots may be rotated over 90 degrees in order to save plotting paper or not. Possible values:

- 0 Rotation is allowed.
- 1 No rotation is allowed.
- 2 Plots are always rotated.

21.14 Common CACTL

Description

Common block CACTL contains element dependent information for the various element subroutines. cactl is used to transport information from main subroutines to element subroutines

Declaration

```
 integer ielem, itype, ielgrp, inpelm, icount, ifirst,
+        notmat, notvec, irelem, nusol, nelem, npoint
 common /cactl/ ielem, itype, ielgrp, inpelm, icount, ifirst,
+               notmat, notvec, irelem, nusol, nelem, npoint
```

Parameters

icount Number of degrees of freedom in element.

ielem Element sequence number.

ielgrp Element group sequence number.

ifirst Indicator if the element subroutine is called for the first time in a series (0) or not (1).

inpelm Number of nodes in element.

irelem Relative element number with respect to element group number ielgrp.

itype Type number of element.

nelem Number of elements in the mesh.

notmat This parameter indicates whether an element matrix is equal to zero (NOTMAT=1) or not (NOTMAT=0) for all elements with standard element sequence number IELGRP. When the user sets NOTMAT equal to 1, this means that the element matrices with element sequence number IELGRP are not added to the large matrix.

notvc See NOTMAT, however with respect to an element vector.

npoint Number of nodes in the mesh.

nusol Number of degrees of freedom in the mesh.

21.15 Common CPLAF

Description

Common block CPLAF is meant for communication between the low-level plotting routines.

Declaration

```
 integer idirec, jtest, ioutsd, dcolour
 double precision wid, ale, xfct, yfct, xprec, yprec, xmint, xmaxt
 double precision ymint, ymaxt, leftm, botm, rightm, topm, aleng,
+                 width
 common /cplaf/ wid, ale, xfct, yfct, xprec, yprec, xmint, xmaxt,
+               ymint, ymaxt, leftm, botm, rightm, topm, aleng,
+               width, idirec, jtest, ioutsd, dcolour
```

Parameters

wid,ale Width resp. length of the plot in the vertical resp. horizontal direction in plotting units.

xfct,yfct Multiplication factors in resp. x and y-direction. These factors are such that all coordinates will be transformed into the unit square (0,1) $\times$ (0,1)

xprec,yprec Coordinates of the last point that has been plotted.

xmint,xmaxt,ymint,ymaxt Window of coordinates to be plotted. All points with (x,y) within the rectangle (xmint,xmaxt) $\times$ (ymint,ymaxt) are plotted.

leftm,botm,rightm,topm Margins in cm around plotting area.

aleng Length of plot in cms.

width Width of plot in cms.

idirec Direction of plotting used. Possible values:

- 1 Normal direction
- -1 Rotated over 90 degrees

jtest Test parameter for the window. Possibilities:

- 0 No tests performed.
- 1 Test if point are within rectangle.

ioutsd Used when JTEST = 1. Meanings are:

- 0 Both new and preceding point are within rectangle.
- 1 New point is within, preceding outside.
- 2 New is outside, preceding inside.
- 3 Both new and preceding point are outside rectangle.

dcolour Default (current) colour to draw lines and text in. If 0 do not use any colour handling.

21.16 Common CONSTA

Description

Common block CONSTA contains some general double precision constants.

Declaration

```
double precision consar
common /consta/ consar(20)
```

Parameters

consar The constants are stored in array CONSAR. Contents:

Pos. 1 pi

Pos. 2 ln(10)

Pos. 3 Default length of plot (Machine dependent).

Pos. 4-20 Not yet defined.

21.17 Common CMACHN

Description

Common block CMACHN is used as common block for determination of computer system as well as some machine-dependent quantities and references.

Declaration

```
 integer  machin, lenwor, numchr, nrec4, iref10, iref73, iref74,
+         irefpl, iplotf, icarcn, iref17, isite, idumma(17)
 common /cmachn/  machin, lenwor, nrec4, numchr, iref10, iref73,
+                 iref74, irefpl, iplotf, icarcn, iref17, isite,
+                 idumma
```

Parameters

icarcn Indicates if the computer uses the carriage control parameter for output to a file (1) or not (0).
This parameter is made equal to the parameter *carriage* in the sepran.env file (Users Manual Section 1.5).

idumma Dummy array, not yet used.

iref10 Dataset reference number with respect to file meshoutput If negative the file is unformatted.
This parameter is made equal to the parameter *unit number for mesh output file* in combination with the parameter *file 10* in the sepran.env file.

iref17 Dataset reference number with respect to file menumsg.
This parameter is made equal to the parameter *unit number for file containing menus* in the sepran.env file.

iref73 Dataset reference number with respect to file sepcomp.inf.
This parameter is made equal to the parameter *unit number for file sepcomp.inf* in the sepran.env file.

iref74 Dataset reference number with respect to file sepcomp.out. If negative the file is unformatted.
This parameter is made equal to the parameter *unit number for file sepcomp.out* in combination with the parameter *file 74* in the sepran.env file.

irefpl Dataset reference number with respect to SEPRAN plot file. If negative the file is unformatted.
This parameter is made equal to the parameter *unit number for default SEPRAN plot file* in combination with the parameter *Default SEPRAN plot file* in the sepran.env file.

iplotf Indicates if a copy of the plot must be made to a file (1) or not (0).

isite Sequence number of site to call special site adapted subroutines.
This parameter is made equal to the parameter *isite* in the sepran.env file.
Possible values:

0 general

1 Eindhoven University of Technology

lenwor This parameter is used in the open statement of direct access files. The record length in the statement rec = ... is given by lenwor times the number of words So if the machine requires that the length is given in words then lenwor should be one. If the record length must be given in bytes, lenwor gives the number of bytes per word.
This parameter is made equal to the parameter *lenwor* in the sepran.env file.

machin Type of computer system.
This parameter is made equal to the parameter *type of computer* in the sepran.env file.
Possible values:

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

nrec4 Number of words that can be stored in one record of the error message file.
This parameter is made equal to the parameter *record length for scratch file* in the sepran.env file.

numchr Number of characters that can be stored in a word
This parameter is made equal to the parameter *number of characters in a word* in the sepran.env file.

21.18 Common CMACHT

Description

Common block CMACHT contains computer (machine) and terminal information in character and strings.

Declaration

```
 character*1  null, escape, formfd, mgraph, malpha, mvt100
 character*2  posdev
 character*16 termin, macnam, usrnam, stdate, sttime, stoper,
+             devdis, spexec, spappn, devhcp, pidsep, dumtxt(8)
 character * 50 sphome, sprenv, namef1, namef2, namef3, namef4,
+               namepl, name10, name73, name74, namere, namewr,
+               spvers, namebm, namems, sepwind, dumtx1(7)
 common /cmacht/ null, escape, formfd, mgraph, malpha, mvt100,
+                posdev, termin, macnam, usrnam, stdate, sttime,
+                stoper, devdis, spexec, spappn, devhcp, pidsep,
+                dumtxt, sphome, sprenv, namef1, namef2, namef3,
+                namef4, namepl, name10, name73, name74, namere,
+                namewr, spvers, namebm, namems, sepwind, dumtx1
```

Parameters

devdis Device for interactive plotting

devhcp Device for hard copies of plot

dumtx1 Dummies for later use

dumtxt Dummies for later use

escape Prefix for many terminal control commands

formfd Form-feed character, new-page on some terminals

malpha Switches Tektronix to alphanumerical mode

macnam Hostname of computer on which the program runs

mgraph Switches Tektronix to graphics mode

mvt100 Switches MS-Kermit back to VT100 mode

name10 Name of file 10 (mesh output file)

name73 Name of file 73 (sepcomp.inf)

name74 Name of file 74 (sepcomp.out)

namebm Name of bitmap file if a bitmap must be made, otherwise blank

namef1 Name of file 1 (information about standard elements)

namef2 Name of file 2 (SEPRAN backing storage file)

namef3 Name of file 3 (scratch file)

namef4 Name of file 4 (error messages)

namems Name of file 17 (menu texts)

namepl Name of default SEPRAN plot file

null Terminator for C-like strings (used in StarBase)

pidsep Process id (Unix only)

posdev Device designation of device to send plot output to. Possible values:

a	Apollo code
c	Calcomp code
cg	CGI code
eu	Eindhoven University, special case
fb	Write input parameters in binary form to file 8
fa	Write input parameters in ascii form to file 8
f1	Write formatted, make only 1 output file
g	GKS code
gi	GKS code (choose your workstation)
hh	StarBase code for HP-display, Colour HiRes (1280x1024)
hl	HP-display, Colour LowRes (1024x768)
hm	HP-display, Monochrome LowRes (1024x768)
ht	HP-display, Turbo SRX (1280x1024, 24 bitplanes)
hx	Starbase on X11
ow	Local version, not supported by SEPRAN
p	HP-GL plotter A4 format (file output)
pb	HP-GL plotter A3
ps	PostScript (greyscale) in ascii to file irefpl (portrait mode)
pl	PostScript (greyscale landscape)
pc	PostScript COLOUR (portrait)
pw	PostScript (colour landscape)
t	Tektronix 4010 code
tk	Tektronix 4010 code, using MS-Kermit emulation
tp	Tektronix 4010 code, using Plot-10 library
tt	Tektronix 4010 code, using Telnet emulation
v	PC, running FTN77/386 with VGA or EGA card
xg	xgks

sepwind Name of window for X11

spappn Indication if the output must be appended to an existing output file (yes) or not (no)

spexec Name of executable

sphome SEPRAN home directory

sprenv name of SEPRAN environment file (Usually sepran.env)

spvers SEPRAN version name

stdate Starting date of program

stoper Type of operating system

sttime Starting time of program (i.e. time at which the program has been started)

termin Terminal type used, full name according to O/S

usrnam Login name of user

All information that is available in the sepran.env file (See Users Manual Section 1.5 is extracted from this file and put into CMACHT by the standard SEPRAN starting subroutines.

21.19 Common CSEPCM

Description

Common block CSEPCM contains a number of texts that can be used in all SEPRAN programs.

Declaration

```
integer ntexts
parameter (ntexts=1000)
character * 80 texts(ntexts)
common /csepcm/ texts
```

Parameters

ntexts length of array texts

texts Array with texts.

The next free position in array TEXTS can be found in the first position of common block CSEPINT (21.20).

21.20 Common CSEPINT

Description

Common block CSEPINT contains a number of integers that can be used in all SEPRAN programs.

Declaration

```
integer nextxt, intdummy
common /csepint/ nextxt, intdummy(999)
```

Parameters

nextxt Next free position in array TEXTS of common block CSEPCM (21.19)

intdummy Dummy integer array, reserved for future purposes.

21.21 Common CGENCONST

Description

Common block CGENCONST contains a number of constants that can be used in all SEPRAN programs. These constants get either a default value or may be initialized by using the input part `GENERAL_CONSTANTS` as described in Section 1.4 of the Users Manual.

Declaration

```
      double precision accuracy_obstacle, genconsts
      integer intgenconst
      common /cgenconst/ accuracy_obstacle, genconsts(999),
     +                   intgenconst(1000)
      save /cgenconst/
```

Parameters

accuracy_obstacle Accuracy parameter with respect to obstacles, see UM Section 1.7.

genconsts Dummy integer array, reserved for future purposes.

intgenconst Dummy integer array, reserved for future purposes.

21.22 Common CDEBUG_ROUTS

Description

Common block CDEBUG_ROUTS contains a number of parameters that activates certain debugging statements in SEPRAN subroutines. These parameters may be set to `true` by using the input part `DEBUG_PARAMETERS` as described in Section 1.4 of the Users Manual. Default they have the value `false`.

Declaration

```
 logical debug_mshsortel3d, debug_mshcrt, debug_mshfillkmeshad,
+        debug_mshsrf, debug_pltrac, debug_buildbf,
+        debug_prob01crobst, debug_probdfbf, debug_probflkprobaa,
+        debug_prorea, debug_prinmtbf, debug_fvcvdf,
+        debug_fvincnd, debug_fvpres, debug_fvprmom,
+        debug_fvsbstep, debug_fvstart,
+        debug_fvtstep, debug_fvviscmatiq2,
+        debug_fvviscmatiq2s, debug_compstat01, debug_extfreesubr,
+        debug_hollowmesh, debug_initfree, plotobst, printobst,
+        plotskipel, printskipel, debug_mshobsfl,
+        debug_mshobstmarkpt, debug_probskippoint, checksign,
+        debug_mshobstmesh, debug_erintpol, check_hanging_nodes,
+        plotedges, trace, checksubtetrahedrons, write_geomview,
+        debug_array
 common /cdebug_routs/ debug_mshsortel3d, debug_mshcrt,
+        debug_mshfillkmeshad,
+        debug_mshsrf, debug_pltrac, debug_buildbf,
+        debug_prob01crobst, debug_probdfbf, debug_probflkprobaa,
+        debug_prorea, debug_prinmtbf, debug_fvcvdf,
+        debug_fvincnd, debug_fvpres, debug_fvprmom,
+        debug_fvsbstep, debug_fvstart,
+        debug_fvtstep, debug_fvviscmatiq2,
+        debug_fvviscmatiq2s, debug_compstat01, debug_extfreesubr,
+        debug_hollowmesh, debug_initfree, plotobst, printobst,
+        plotskipel, printskipel, debug_mshobsfl,
+        debug_mshobstmarkpt, debug_probskippoint, checksign,
+        debug_mshobstmesh, debug_erintpol, check_hanging_nodes,
+        plotedges, trace, checksubtetrahedrons, write_geomview,
+        debug_array(986)
 save /cdebug_routs/
```

Parameters

debug_name Activates debugging in a subroutine with the name `name`. See Section 1.3

plotobst Activates making of plots with respect to obstacles

printobst Activates printing of information with respect to obstacles

debug_array Dummy integer array, reserved for future purposes.

22 Memory management

22.1 Introduction

In this chapter the SEPRAN memory management subroutines are treated.
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 SEPRAN 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 this chapter we deal with the following paragraphs:

22.2 describes the tasks of the memory management system.

22.3 gives an overview of the memory management tools.

22.4 shows how to use the memory management system.

22.5 describes the structure of the buffer array IBUFFR.

22.6 treats the various tools.

22.2 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) 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.

22.3 An overview of the memory management tools

The following memory management tools are available:

Section	Tool	Description
22.6.1	ini070	Activate old permanent array
22.6.2	ini061	Activate old solution array and corresponding arrays
22.6.3	ini091	Create or activate new permanent integer array
22.6.4	ini090	Create or activate new permanent double precision array
22.6.5	ini086	Create or activate solution array and corresponding arrays
22.6.6	ini094	Extend a permanent integer array
22.6.7	ini092	Extend a permanent double precision array
22.6.8	ini093	Make a temporary integer array permanent
22.6.9	ini095	Make a temporary double precision array permanent
22.6.10	ini072	Create space for a temporary array
22.6.11	ini076	Delete permanent array
22.6.12	ini079	reserve space at the end of the buffer and decrease active length temporarily
22.6.13	inirst	Reset buffer length to old value, i.e. remove array created by ini079
22.6.14	inipri	Decrease priority
22.6.15	inispr	Set priority of an array explicitly
22.6.16	iniget	Get pointer of an array in IBUFFR
22.6.17	inidgt	Get pointer of an array in BUFFER
22.6.18	inilen	Get length of an array, expressed in integers
22.6.19	inidln	Get length of an array, expressed in reals
22.6.20	inichk	Check memory management if debug option is used
22.6.21	inisgl	Transform pointer for BUFFER into pointer for IBUFFR
22.6.22	inidbl	Transform pointer for IBUFFR into pointer for BUFFER
22.6.23	inignm	Return with name of array stored in IBUFFR
22.6.24	inigpr	Return with priority of array stored in IBUFFR
22.6.25	inimvc	Return with highest sequence number of array stored in IBUFFR
22.6.26	inistr	Find structure of solution vector with exactly NUMBER unknowns per point

22.4 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 in blank common. Next IBUFFR must be initialized using the starting subroutines as described in Chapter 4. New permanent or temporary arrays may be created in main SEPRAN 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. 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 SEPRAN subroutine the integer array "array" is addressed by IBUFFR(IParray), in the actual SEPRAN subroutine the array gets its usual array name and corresponding declaration, i.e. real, double precision or integer.

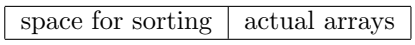
So a typical main SEPRAN main program and subroutines may look like:

```
      program sepran_main
      implicit none
      integer IBUFFR, nbuffr
      parameter ( nbuffr=1000000 )
      common IBUFFR(nbuffr)
      call sepran_main_sub ( nbuffr )
      end

      subroutine sepran_main_sub ( nbuffr )
      implicit none
      integer nbuffr

      integer kmesh(...), kprob(...), intmat(5), iinstr(3)

c     --- Start memory manager

      iinstr(1) = ...
      iinstr(2) = ...
      iinstr(3) = nbuffr
      call sepstn ( kmesh, kprob, intmat, iinstr )

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 ini070 ( isar1 )
      isar2 = 0
      lenar2 = ....
      call ini090 ( isar2, lenar2, 'name_of array 2' )
      lentmp = ....
      call ini072 ( lentmp, iptmp )

c     --- Get pointers, except from temporary array

      ipiar1 = iniget ( isar1 )
      ipiar2 = inidgt ( isar2 )

c     --- Call actual subroutine

      call subr1 ( IBUFFR(ipar1), BUFFER(ipar2), IBUFFR(iptmp) )

c     --- decrease priorities

      call inipri

      end

c     --- Actual subroutine subr1

      subroutine subr1 ( iarray, rarray, iwork )
      implicit none
      integer iarray(*), iwork(*)
      double precision rarray(*)

c     --- actual statements

      end
```

22.5 The structure of array IBUFFR

Array IBUFFR consists of two parts: the actual arrays and some space needed for sorting. The global structure is:



The space for sorting is at this moment 1500 positions.

The administration of ibuffr is stored in common block CARRAY:

```
integer iinfor, infor
common /carray/ iinfor, infor(3,1500)
```

Parameters

iinfor Number of arrays that are stored in IBUFFR. At most 1500 arrays can be stored, hence $0 \leq$ IINFOR ≤ 1500.

infor Information concerning the storage of arrays in IBUFFR is stored in the following way:

Pos. (1,i) If > 0: array with sequence number i is stored in IBUFFR from position infor(1,i). If < 0: array i is stored in file 3 from record number -infor(1,i)

Pos. (2,i) Length of the array in words (integers).

Pos. (3,i) Priority number. Is used to decide which arrays are put to file 3. $0 \leq$ prio ≤ 10. Arrays with priority 100 are always kept in-core

Each array that is stored in IBUFFR gets a sequence number (i). This sequence number refers to array INFOR and must a number between 1 and IINFOR. The starting address of an array is stored in INFOR(1,i), provided the array is stored in-core. If the array has been swapped INFOR(1,i) gets a negative value.

22.6 Detailed description of the memory management tools

22.6.1 Subroutine INI070

Description

General memory management subroutine that is used to activate an old permanent array. This array must have been created before by a mm subroutine.
Activation implies the following steps:

- If the array is available in IBUFFR, then the priority is set equal to 10, and the subroutine returns
- When the array is only available on backing storage, and not in IBUFFR then the array is read from file 3 and space is reserved in array IBUFFR
- If no space is available in array IBUFFR, INI070 tries to create space in the following ways:
  - by writing low priority arrays to backing storage
  - if this is not sufficient by compressing array ibuffr after writing of all arrays with priority smaller than 10 to backing storage

Heading

```
subroutine ini070 ( index )
```

Parameters

INTEGER INDEX

INDEX must contain the sequence number of the existing array ($0 < INDEX \leq 1500$). The creation mm subroutine has initialized INDEX before.

Input

INDEX must have a value.

Output

It is guaranteed that the array corresponding to the sequence number INDEX is positioned in IBUFFR. See subroutine INI072 (22.6.10)

Remarks

- Starting addresses of other arrays in IBUFFR may have been changed by the call of INI070, and hence all starting addresses to be used in the subroutine must be recomputed.
- Subroutine INI070 is in fact a call to subroutine INI060:

  ```
  subroutine ini060 ( ibuffr, index )
  ```

  Hence one may replace INI070 by INI060, provided ibuffr is available.

22.6.2 Subroutine INI061

Description

General memory management subroutine that is used to activate an old solution array as well as the corresponding arrays KPROB part f, h and p if they exist. This solution array must have been created before by a mm subroutine.
Activation means in this case exactly the same as for subroutine INI070 22.6.1.

Heading

```
subroutine ini061 ( ibuffr, isol, kprob, indprf, indprp, nunkp, nphys )
```

Parameters

INTEGER IBUFFR(∗), ISOL(∗), KPROB(∗), INDPRF, INDPRP, NUNKP, NPHYS

IBUFFR Standard SEPRAN buffer array as described in Section 22.4. Usually IBUFFR is stored in blank common.

ISOL Standard SEPRAN array of length 5 containing information of the storage of a solution vector or vector of special structure stored in the buffer array IBUFFR. ISOL must have been created before

KPROB Standard SEPRAN array containing information about the problem description. KPROB must have been filled before for example by a SEPRAN start routine that reads the mesh.

INDPRF At output INDPRF contains information concerning the KPROB part f corresponding to ISOL.
If no KPROB part f corresponds to ISOL, INDPRF = 0, otherwise INDPRF contains the sequence number of array KPROB part f with respect to IBUFFR.

INDPRH At output INDPRH contains information concerning the KPROB part h corresponding to ISOL.
If no KPROB part h corresponds to ISOL, INDPRH = 0, otherwise INDPRH contains the sequence number of array KPROB part h with respect to IBUFFR.

INDPRP At output INDPRP contains information concerning the KPROB part p corresponding to ISOL.
If no KPROB part p corresponds to ISOL, INDPRP = 0, otherwise INDPRP contains the sequence number of array KPROB part p with respect to IBUFFR.

NUNKP At output NUNKP contains the maximal number of degrees of freedom per point.

NPHYS At output NPHYS contains the maximal number of degrees of freedom per point or the number of physical variables if defined.
If ISOL is a vector of special structure and the number of degrees of freedom per point is not constant then NPHYS = 0.
If ISOL is a vector of type 116 (see Section 22.4), NPHYS contains the number of degrees of freedom per element.

Input

The arrays ISOL and KPROB must have been filled before.

Output

The parameters INDPRF, INDPRP, NUNKP and NPHYS have got a value.
The arrays corresponding to ISOL and possibly KPROB parts f and p have been activated. This means that these arrays are positioned in-core in the buffer array IBUFFR.

Remark

Starting addresses of other arrays in IBUFFR may have been changed by the call of INI061, and hence all starting addresses to be used in the subroutine must be recomputed.

22.6.3 Subroutine INI091

Description

General memory management subroutine that is used to create a new integer SEPRAN array in array IBUFFR. If the array already exists INI091 activates the array if it is positioned already in IBUFFR. However, if the array is only positioned at backing storage, the values are not read and actually a new array is created using the old sequence number. In that case the old array is not available anymore.
If the array did not exist and there is not enough contiguous space in array IBUFFR, space is reserved in exactly the same way as is done by INI070.
If the array did exist, but the length of the previous array is smaller than that of the new one, the old sequence number is used, but a new array is created. As a consequence the old array is not available anymore.
The array created gets automatically priority 10.

Heading

```
subroutine ini091 ( index, length, namear )
```

Parameters

INTEGER INDEX, LENGTH

CHARACTER $*$ **(**$*$**)** NAMEAR

INDEX The value of INDEX is the sequence number of the array to be created. INDEX may be both input and output parameter. It is therefore absolutely necessary that INDEX is a variable and not a constant.
In many SEPRAN applications INDEX is a position in a SEPRAN array, for example ISOL(1) or KMESH(17).
The value of INDEX at input indicates if the array to be created existed before or not. If INDEX is in the range 1 to IINFOR, with IINFOR the number of arrays stored in IBUFFR, it is assumed that the array did exist before and the old positions are used if possible. Anyway the old sequence number is used and the array corresponding to this sequence number may be lost. Since in FORTRAN programs variables are not initialized automatically, it is possible that INDEX has by some coincidence, a value in the range 1 to IINFOR without the user wanting it. So if the array to be created must be new, be sure that you initialize INDEX to 0.

LENGTH gives the length of the array to be created in integers.

NAMEAR must contain the name of the array. This must be a string, for example the name of the array between quotes.
This name is only used for output purposes and possibly error messages. So the name may be chosen in such a way that it is clear for the user what is identified. Only the first 9 characters are used.

Input

INDEX, LENGTH and NAMEAR must have a value.

Output

The value of INDEX may have been computed and possibly changed.
Space in IBUFFR has been reserved for the new array.

Remarks

- Starting addresses of other arrays in IBUFFR may have been changed by the call of INI091, and hence all starting addresses to be used in the subroutine must be recomputed.
- Subroutine INI091 is in fact a call to subroutine INI081:

  ```
  subroutine ini081 ( ibuffr, index, length, namear )
  ```

  Hence one may replace INI091 by INI081, provided ibuffr is available.

22.6.4 Subroutine INI090

Description

General memory management subroutine that is used to create a new real (double precision) SEPRAN array in array BUFFER. In fact INI090 is identical to INI091 (22.6.3) with the exception that double precision arrays are created.

Heading

```
subroutine ini090 ( index, length, namear )
```

Parameters

INTEGER INDEX, LENGTH

CHARACTER $*$ $(*)$ NAMEAR

INDEX See subroutine INI091

LENGTH gives the length of the array to be created in (double precision) reals.

NAMEAR See subroutine INI091

Input

INDEX, LENGTH and NAMEAR must have a value.

Output

The value of INDEX may have been computed and possibly changed.
Space in IBUFFR has been reserved for the new array.

Remarks

- Starting addresses of other arrays in IBUFFR may have been changed by the call of INI090, and hence all starting addresses to be used in the subroutine must be recomputed.
- Subroutine INI090 is in fact a call to subroutine INI080:

  ```
  subroutine ini080 ( ibuffr, , length, namear )
  ```

  Hence one may replace INI090 by INI080, provided ibuffr is available.

22.6.5 Subroutine INI086

Description

General memory management subroutine that is used to create a new solution array or array of special structure in array BUFFER. In fact INI086 has exactly the same task as INI090 (22.6.4), however, the user does not have to give the length of the array explicitly.
Besides that, the arrays KPROB parts f and p are activated, provided they exist. Compare with subroutine INI061 (22.6.2) for this matter.

Heading

```
subroutine ini086 ( ibuffr, isol, itypvc, ivec, iprob, kmesh, kprob,
                    indprf, indprp, nphys, namear )
```

Parameters

INTEGER ISOL(5), ITYPVC, IVEC, IPROB, KMESH(∗), KPROB(∗), INDPRF, INDPRP, NPHYS

CHARACTER ∗ **(∗)** NAMEAR

ISOL Standard SEPRAN array of length 5 in which the information of the storage of a solution vector or vector of special structure stored in the buffer array BUFFER will be stored. If ISOL has been filled the old positions in BUFFER will be used under the same conditions as for subroutine INI090.

ITYPVC Defines the type of SEPRAN vector to be created. The following values of ITYPVC are possible:

110 The array to be created is of the type of a solution vector.

115 The array to be created is of the type of a vector of special structure.

116 The array to be created is of the type of a vector of special structure, defined per element.

IVEC This parameter is only used for arrays of special structure (115 or 116).
If ITYPVC = 115 IVEC defines the type of the array,
if ITYPVC = 116 IVEC defines the number of degrees of freedom per element.

IPROB defines the problem sequence number to which the array to be created belongs.

KMESH Standard SEPRAN array containing information about the mesh. KMESH must have been filled before by a mesh generation subroutine or by a subroutine that reads the mesh.

KPROB Standard SEPRAN array containing information about the problem description. KPROB must have been filled before for example by a SEPRAN start routine that reads the mesh.

INDPRF At output INDPRF contains information concerning the KPROB part f corresponding to ISOL.
If no KPROB part f corresponds to ISOL, INDPRF = 0, otherwise INDPRF contains the sequence number of array KPROB part f with respect to IBUFFR.

INDPRP At output INDPRP contains information concerning the KPROB part p corresponding to ISOL.
If no KPROB part p corresponds to ISOL, INDPRP = 0, otherwise INDPRP contains the sequence number of array KPROB part p with respect to IBUFFR.

NPHYS At output NPHYS contains the maximal number of degrees of freedom per point or the number of physical variables if defined.
If ISOL is a vector of special structure and the number of degrees of freedom per point is not constant then NPHYS = 0.
If ISOL is a vector of type 116 (see Section 22.4), NPHYS contains the number of degrees of freedom per element.

Input

ITYPVC, IVEC, IPROB and NAMEAR must have a value.
The arrays KPROB and KMESH must have been filled.
ISOL(1) must have been initialized, for example with 0.

Output

Array ISOL has been filled.
The parameters INDPRF, INDPRH, INDPRP and NPHYS have got a value.
The arrays corresponding to ISOL has been created. If existing, the arrays corresponding to INDPRF, INDPRH and INDPRP have been activated. This means that these arrays are positioned in-core in the buffer array IBUFFR.

Remark

Starting addresses of other arrays in IBUFFR may have been changed by the call of INI086, and hence all starting addresses to be used in the subroutine must be recomputed.

22.6.6 Subroutine INI094

Description

Extend an existing integer array. If no space can be found following this array new positions are created and the old contents are copied to the new positions.
All remarks concerning the creation of arrays given in 22.6.3 are still valid.

Heading

```
subroutine ini094 ( index, length, namear )
```

Parameters

INTEGER INDEX, LENGTH

CHARACTER $*$ $(*)$ NAMEAR

INDEX The value of INDEX is the sequence number of the array to be extended.

LENGTH gives the length of the array to be created in integers.

NAMEAR must contain the name of the array. See 22.6.3.

Input

LENGTH, and NAMEAR must have a value.
INDEX must have been initialized.

Output

Space in IBUFFR has been reserved for the new array.

Remarks

- Starting addresses of other arrays in IBUFFR may have been changed by the call of INI094, and hence all starting addresses to be used in the subroutine must be recomputed.
- Subroutine INI094 is in fact a call to subroutine INI084:

  ```
  subroutine ini084 ( ibuffr, index, length, namear )
  ```

  Hence one may replace INI094 by INI084, provided ibuffr is available.

22.6.7 Subroutine INI092

Description

Extend a real (double precision) existing array in BUFFER. In fact INI092 is identical to INI094 (22.6.6), however, with integers replaced by (double precision) reals.

Heading

```
subroutine ini092 ( index, length, namear )
```

Parameters

INTEGER INDEX, LENGTH

CHARACTER $*$ $(*)$ NAMEAR

INDEX See INI094 (22.6.6).

LENGTH gives the length of the array to be created in double precision reals.

NAMEAR must contain the name of the array. See 22.6.3.

Input

LENGTH, and NAMEAR must have a value.
INDEX must have been initialized.

Output

Space in IBUFFR has been reserved for the new array.

Remarks

- Starting addresses of other arrays in IBUFFR may have been changed by the call of INI092, and hence all starting addresses to be used in the subroutine must be recomputed.
- Subroutine INI092 is in fact a call to subroutine INI082:

  ```
  subroutine ini082 ( ibuffr, index, length, ipcopy, lencop, namear )
  ```

  Hence one may replace INI092 by INI082, provided ibuffr is available.

22.6.8 Subroutine INI093

Description

General memory management subroutine which reserves space for a new SEPRAN array in blank common and, moreover, simplifies the computation of the starting address of the array in blank common.
The difference with subroutine INI091 (22.6.3) is, that (now) it is assumed that the array has already been constructed in IBUFFR from the temporary position IPCOPY, which implies that there is already space enough in IBUFFR. Furthermore the array is copied from the temporary positions to the final positions.
If the array did not exist, or the space allocated to the old array is insufficient to store the new one, then space in array IBUFFR is reserved. If no space is available in array IBUFFR, INI093 starts the new array from position IPCOPY.

Heading

```
subroutine ini093 ( index, length, ipcopy, namear )
```

Parameters

INTEGER INDEX, LENGTH, IPCOPY

CHARACTER $*$ $(*)$ NAMEAR

INDEX See INI094 (22.6.6)

LENGTH gives the length of the array to be created in integers.

IPCOPY Temporary address in IBUFFR, from which the contents of the array have been stored.

NAMEAR must contain the name of the array. See 22.6.3.

Input

LENGTH, IPCOPY and NAMEAR must have a value.
INDEX must have been initialized.

Output

The value of INDEX may have been computed and possibly changed.
Space in IBUFFR has been reserved for the new array.

Remarks

- Starting addresses of other arrays in IBUFFR may have been changed by the call of INI093, and hence all starting addresses to be used in the subroutine must be recomputed.
- Subroutine INI093 is in fact a call to subroutine INI083:

  ```
  subroutine ini083 ( ibuffr, index, length, ipcopy, namear )
  ```

  Hence one may replace INI093 by INI083, provided ibuffr is available.

22.6.9 Subroutine INI095

Description

This subroutine is exactly equal to subroutine INI093 (26.6.8), with the exception that the starting address and the length refer to array BUFFER instead of IBUFFR.
In case of a single precision machine (for example a 64 bits CRAY), there is no difference with INI093, in other cases the addresses and length are adapted.

Heading

```
subroutine ini095 ( index, length, ipcopy, namear )
```

Parameters

INTEGER INDEX, LENGTH, IPCOPY

CHARACTER $*$ **(**$*$**)** NAMEAR

INDEX See INI094 (22.6.6)

LENGTH gives the length of the array to be created in double precision reals.

IPCOPY Temporary address in BUFFER, from which the contents of the array have been stored.

NAMEAR must contain the name of the array. See 22.6.3.

Input

LENGTH, IPCOPY and NAMEAR must have a value.
INDEX must have been initialized.

Output

The value of INDEX may have been computed and possibly changed.
Space in IBUFFR has been reserved for the new array.

Remarks

- Starting addresses of other arrays in IBUFFR may have been changed by the call of INI095, and hence all starting addresses to be used in the subroutine must be recomputed.
- Subroutine INI095 is in fact a call to subroutine INI085:

  ```
  subroutine ini085 ( ibuffr, index, length, ipcopy, namear )
  ```

  Hence one may replace INI095 by INI085, provided ibuffr is available.

22.6.10 Subroutine INI072

Description

General memory management subroutine to create temporary space in array IBUFFR in blank common.
This subroutine creates space and computes the starting position of a temporary array of length LENGTH in array IBUFFR.
If space is available IBFREE is not adjusted, which means that INI072 may only be called once during a process because otherwise the same starting address might be found. If space for more arrays must be reserved at one time it is necessary to give LENGTH the value of the sum of the length of all these arrays.
INI072 must always be called after all calls of the subroutines INI070 and INI091. A call of INI070 or INI091 after the call of INI072 may have undesirable effects.
Remember: it is necessary that first the reals must be positioned in the temporary space and then the integers. This is due to the fact that reals should start add odd addresses in IBUFFR.
If no space is available in array IBUFFR, INI072 tries to create space in the following ways:

- by writing low priority arrays to backing storage
- if this is not sufficient by compressing array ibuffr after writing of all arrays with priority smaller than 10 to backing storage

As a consequence all starting positions that have been computed before, may have been changed.

Heading

```
subroutine ini072 ( length, ipstar )
```

Parameters

INTEGER LENGTH, IPSTAR

LENGTH Length of the space to be reserved in words.

IPSTAR At output IPSTAR contains the starting position of the temporary array in array IBUFFR.

Input

LENGTH must have a value.

Output

IPSTAR has got a value.
Space has been reserved for the temporary array.

Remark

Starting addresses of other arrays in IBUFFR may have been changed by the call of INI072, and hence all starting addresses to be used in the subroutine must be recomputed.

22.6.11 Subroutine INI076

Description

General memory management subroutine to delete permanent arrays from the SEPRAN memory. The array is removed from common block CARRAY and its space can be reused.

Heading

```
subroutine INI076 ( INDEX )
```

Parameters

INTEGER INDEX

INDEX The value of index is the sequence number of the array that must be deleted.

Input

INDEX must have a value.

Output

The administration has been updated.

22.6.12 Subroutine INI079

Description

Reserve temporarily LENGTH words at the end of array IBUFFR. The starting position is equal to IPSTAR and NBUFFR is set equal to IPSTAR-1. NBUFOL gets the old value of NBUFFR, so that at the end NBUFFR can be reset. If the last positions of IBUFFR are already occupied, the corresponding arrays are written to file 3 and in the case these arrays have priority 10 or larger, they are read in new positions in IBUFFR
The main reason for using INI079 is that memory management subroutines may change the contents of IBUFFR for example by moving parts to other positions. If a part of IBUFFR is used as input array for a subroutine calling mm subroutines the effect may be that the input is actually destroyed. To prevent this the input must be put into a part of IBUFFR that is not changed by the mm subroutines. Subroutine INI079 may be of help to achieve this since only parts in IBUFFR with addresses smaller than NBUFFR may be changed.
So a natural way to use INI079 is the following one:

- create sufficient space by INI079
- copy the contents of the array that may not be changed to the new starting address
- call the specific subroutine using this new start address for the array
- reset NBUFFR by INIRST

Heading

```
subroutine INI079 ( IPSTAR, LENGTH, NBUFOL )
```

Parameters

INTEGER LENGTH, IPSTAR, NBUFOL

LENGTH Length of the space to be reserved in words.

IPSTAR At output IPSTAR contains the starting position of the temporary array in array IBUFFR.

NBUFOL Value of NBUFFR at input.
This value may be used to reset NBUFFR in a later stage.

Input

LENGTH must have a value.

Output

IPSTAR and NBUFOL have got a value.
Space has been reserved for the temporary array.

Remark

Starting addresses of other arrays in IBUFFR may have been changed by the call of INI079, and hence all starting addresses to be used in the subroutine must be recomputed.

22.6.13 Subroutine INIRST

Description

Reset buffer length to old value. In fact this subroutine removes the arrays created by ini079.

Heading

```
subroutine INIRST ( NBUFOL )
```

Parameters

INTEGER NBUFOL

NBUFOL Value to which the buffer length must be reset.
NBUFOL has got a value by the call to subroutine INI079. Using the specific value of one specific call to INI079, means that all arrays created by calls to INI079 from this specific one are removed.

Input

NBUFOL should have a value.

Output

The length of the buffer array has been reset to NBUFOL integers.

22.6.14 Subroutine INIPRI

Description

General memory management subroutine that decreases the priorities of all arrays stored in IBUFFR except the ones with priority number 100 or larger.
INIPRI is called by all main SEPRAN subroutines. After the call of INIPRI each array in IBUFFR is a possible candidate to be written to backing storage if space is needed in blank common.
Arrays are written according to their respective priorities.

Heading

```
subroutine INIPRI
```

Output

The priorities of arrays with priority less than 100 are decreased by one.

22.6.15 Subroutine INISPR

Description

set priority of one array stored in IBUFFR explicitly.
This is especially meant to give an array priority number 100, since arrays with priority 100 are always kept in-core.

Heading

```
subroutine INISPR ( INDEX, IPRIO )
```

Parameters

INTEGER INDEX, IPRIO

INDEX must contain the sequence number of the existing array ($0 < INDEX \leq 1500$) for which the priority must be adapted.

IPRIO New priority for the array.

Input

INDEX and IPRIO must have a value.

Output

The priority of array "INDEX" has been adapted.

22.6.16 Subroutine INIGET

Description

This subroutine returns with the pointer with respect to IBUFFR corresponding to the array with sequence number INDEX It is checked if the pointer is available and if the priority of the array is at least 10. If the priority is less than 10, it means that the array has not been activated correctly.
INIGET is a save way to compute the pointer of an array. It is far more reliable than to use array INFOR in common block CARRAY.

Heading

```
IPOINTER = INIGET ( INDEX )
```

Parameters

INTEGER INDEX, INIGET

INDEX must contain the sequence number of the existing array ($0 < INDEX \leq 1500$), for which the starting address is required.

INIGET Contains the starting address of the array in IBUFFR at output. The array may be addressed in a subroutine by

```
    index = ...
    ipointer = iniget ( index )
subroutine subroutine ( ......, ibuffr(ipointer), ... )
```

Input

INDEX must have a value.
The array corresponding to INDEX must have been activated properly.

Output

INIGET has got a value.

Remark

After the call to memory management subroutines, the pointers computed by INIGET may not longer be used, since they may have been changed. In that case a new call to INIGET is necessary.

22.6.17 Subroutine INIDGT

Description

This subroutine returns with the pointer with respect to BUFFER corresponding to the array with sequence number INDEX It is checked if the pointer is available and if the priority of the array is at least 10. If the priority is less than 10, it means that the array has not been activated correctly.
INIDGT is exactly identical to INIGET (22.6.16), however, it computes the starting address with respect to BUFFER instead of IBUFFR.

Heading

```
IPOINTER = INIDGT ( INDEX )
```

Parameters

INTEGER INDEX, INIDGT

INDEX must contain the sequence number of the existing array ($0 < INDEX \leq 1500$), for which the starting address is required.

INIDGT Contains the starting address of the array in BUFFER at output.

Input

INDEX must have a value.
The array corresponding to INDEX must have been activated properly.

Output

INIDGT has got a value.

22.6.18 Subroutine INILEN

Description

This subroutine returns with the length of the array with sequence number INDEX. This length is expressed in words, where an integer is one word and a real is two words at most computers.

Heading

```
LENGTH = INILEN ( INDEX )
```

Parameters

INTEGER INDEX, INILEN

INDEX must contain the sequence number of the existing array ($0 < INDEX \leq 1500$), for which the length is required.

INILEN Contains the length of the array in IBUFFR at output.

Input

INDEX must have a value.

Output

INILEN has got a value.

22.6.19 Subroutine INIDLN

Description

This subroutine returns with the length of the array with sequence number INDEX. This length is expressed in reals instead of words. So INIDLN is the equivalent of INILEN with respect to BUFFER.

Heading

```
LENGTH = INIDLN ( INDEX )
```

Parameters

INTEGER INDEX, INIDLN

INDEX must contain the sequence number of the existing array ($0 < INDEX \leq 1500$), for which the length is required.

INIDLN Contains the length of the array in BUFFER at output.

Input

INDEX must have a value.

Output

INIDLN has got a value.

22.6.20 Subroutine INICHK

Description

This subroutine checks all control integers around arrays in IBUFFR. These control integers are only set if the file sepran.dbg exists and the debug level is at least one. It is a help subroutine to control possible over writing of array bounds in IBUFFR.

Heading

`subroutine INICHK`

Output

The control integers have been checked and in case of a corruption an error message is given.

22.6.21 Subroutine INISGL

Description

This subroutine recomputes a pointer with respect to BUFFER into a pointer with respect to IBUFFR.

Heading

```
IPOINTER_IBUFFR = INISGL ( IPOINTER_BUFFER )
```

Parameters

INTEGER IPOINTER_BUFFER, INISGL

IPOINTER_BUFFER must contain the pointer with respect to BUFFER that must be transformed into a pointer with respect to IBUFFR.

INISGL Contains the recomputed pointer at output.

Input

IPOINTER_BUFFER must have a value.

Output

INISGL has got a value.

22.6.22 Subroutine INIDBL

Description

This subroutine recomputes a pointer with respect to IBUFFR into a pointer with respect to BUFFER.

Heading

```
IPOINTER_BUFFER = INIDBL ( IPOINTER_IBUFFR )
```

Parameters

INTEGER IPOINTER_IBUFFR, INIDBL

IPOINTER_IBUFFR must contain the pointer with respect to IBUFFR that must be transformed into a pointer with respect to BUFFER.

INIDBL Contains the recomputed pointer at output.

Input

IPOINTER_IBUFFR must have a value.

Output

INIDBL has got a value.

22.6.23 Subroutine INIGNM

Description

This subroutine returns with the name of the array with sequence number INDEX.

Heading

```
NAME_ARRAY = INIGNM ( INDEX )
```

Parameters

INTEGER INDEX

CHARACTER $*$ **9** INIGNM

INDEX must contain the sequence number of the existing array ($0 < INDEX \leq 1500$), for which the name is required.

INIGNM Contains the name of the array at output.

Input

INDEX must have a value.

Output

INIGNM has got a value.

22.6.24 Subroutine INIGPR

Description

This subroutine returns with the priority of the array with sequence number INDEX.

Heading

```
IPRIO = INIGPR ( INDEX )
```

Parameters

INTEGER INDEX, INIGPR

INDEX must contain the sequence number of the existing array ($0 < INDEX \leq 1500$), for which the priority is required.

INIGPR Contains the priority of the array at output.

Input

INDEX must have a value.

Output

INIGPR has got a value.

22.6.25 Subroutine INIMVC

Description

This subroutine returns with the highest sequence number of all the arrays stored in IBUFFR.

Heading

```
IHIGH = INIMVC()
```

Output

INIMVC has got a value.

22.6.26 Subroutine INISTR

Description

Find the structure of the vector with exactly NUMBER unknowns per point

Heading

```
subroutine inistr ( number, kprob, itypvc, ivec, iprob )
```

Parameters

INTEGER NUMBER, KPROB(∗), ITYPVC, IVEC, IPROB

NUMBER The number of unknowns per point that is required for the vector.

KPROB Standard SEPRAN array, containing information of the problem.

ITYPVC Indication of the type of solution vector If itypvc = 110 at input then the output vector may be a vector of type 110 or 115 else of type 115 only.
Output:

110 solution vector

115 vector of special structure

IVEC Sequence number of vector of special structure

IPROB Actual problem number.

INTEGER NUMBER, KPROB(∗), ITYPVC, IVEC, IPROB

Input

Array KPROB must have been filled.
The parameters NUMBER, ITYPVC and IPROB must have a value.

Output

The parameters ITYPVC and IVEC have a value.

22.7 An example of the use of the SEPRAN memory management subroutines

In this section we consider how the SEPRAN memory management subroutines may be used in an actual program.

We assume that within an actual SEPRAN program the following actions must be performed:

- Print a solution vector (ISOL1) with exactly one degree of freedom per point, together with the corresponding co-ordinates.
- Print the second degree of freedom of a solution vector (ISOL2) with exactly three degrees of freedom per point.
- Print the third degree of freedom of a solution vector (ISOL3) with a variable number of degrees of freedom per point.
- Print the first degree of freedom of an array of special structure (IVECTR). Manipulate this vector and create a new one. In this way create a new SEPRAN array of special structure number 3 (one degree of freedom per point) and problem sequence number 2. Use NPOINT positions of work space. So we assume that IVEC = 3 has been defined by the call of PROBDF (4.4) or SEPSTR (4.2.1) and that it corresponds to one degree of freedom per point (IPROB=2).

Such a program may be of the following form:

```
      program example
      implicit none
      integer isol1(5), isol2(5), isol3(5), ivectr(5), kmesh(100),
     +        kprob(500), iown(5), ....

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

      integer iinfor, infor
      common /carray/ iinfor, infor(3,1500)

      integer nbuffr, kbuffr, intlen, ibfree
      common /cbuffr/ nbuffr, kbuffr, intlen, ibfree

      integer npoint, nphys, indprf, indprh, indprp, ipwork, nunkp,
     +        ipcoor, ipusol1, ipusol2, ipkprh, ipkprp, ipusol3,
     +        ipown, ipvectr

      integer iniget, inidgt

c     --- Remark: the length of array IBUFFR has already been fixed by
c                 subroutine SEPSTR (\ref{rout-SEPSTR})

      call sepstn ( . . . )
      call presbc ( . . . )
      call linprb ( . . . )
          .
          .
```

```
          .

c     --- Print array USOL1 corresponding to ISOL1
c         The number of nodes is given by: NPOINT
c         The information of the co-ordinates is stored in KMESH(23)
c         Guarantee that the co-ordinates are available in IBUFFR:  INI070
c         Since USOL1 may be renumbered INI078 must be called for ISOL1

      call ini070 ( kmesh(23) )
      call ini078 ( isol1, kprob, indprf, indprh, indprp, nunkp, nphys )
      npoint = kmesh(8)
      ipcoor = inidgt(kmesh(23))
      ipusol1 = inidgt(isol1(1))
      if ( indprh.eq.0 ) then
         ipkprh = 1
      else
         ipkprh = iniget(indprh)
      end if
      call prin1 ( npoint, buffer(ipcoor), buffer(ipusol1),
     +             ibuffr(ipkprh), indprh )

c     --- Print USOL2 corresponding to ISOL2
c         ISOL2 must be activated
c         All pointers must be recomputed

      call ini078 ( isol2, kprob, indprf, indprh, indprp, nunkp, nphys )
      ipusol2 = inidgt(isol2(1))
      ipcoor = inidgt(kmesh(23))
      if ( indprp.eq.0 ) then
         ipkprp = 1
      else
         ipkprp = iniget(indprp)
      end if
      call prin2 ( npoint, buffer(ipcoor), buffer(ipusol2),
     +             ibuffr(ipkprp) )

c     --- Print USOL3 corresponding to ISOL3

      call ini078 ( isol3, kprob, indprf, indprh, indprp, nunkp, nphys )
      ipusol3 = inidgt(isol3(1))
      ipcoor = inidgt(kmesh(23))
      if ( indprp.eq.0 ) then
         ipkprp = 1
      else
         ipkprp = iniget(indprp)
      end if
      call prin3 ( npoint, buffer(ipcoor), buffer(ipusol3),
     +             ibuffr(ipkprp) )

c     --- Print the first degree of freedom of IVECTR
c         Manipulate this vector
c         Store the result in array IOWN of length NPOINT
c         The work space has length NPOINT

      call ini078 ( ivectr, kprob, indprf, indprh, indprp, nunkp,
```

```
     +              nphys )

c     --- Since this program must work for computers with 32 and 64 bits
c         arithmetic, parameter intlen is used to define the length of
c         real work arrays

      call ini090 ( iown(1), npoint, 'iown1' )
      call ini072 ( npoint*intlen, ipwork )
      ipown = inidgt(iown(1))
      ipvectr = inidgt(ivectr(1))
      ipcoor = inidgt(kmesh(23))

      call manipulate ( npoint, buffer(ipcoor), buffer(ipvectr),
     +                  buffer(ipown), buffer(ipwork) )

c     --- Fill information in IOWN

      iown(2) = 115
      iown(3) = 0
      iown(4) = 1003
      iown(5) = npoint
          .
          .
          .
          .
          .
      end

      subroutine prin1 ( npoint, coor, usol, kprobh, indprh )
      implicit none
      integer npoint, kprobh(*), indprh
      double precision coor(2,npoint), usol(npoint), x, y, value

c     --- a two-dimensional mesh is assumed
c         Print USOL

      do 100 i = 1, npoint
         x = coor(1,i)
         y = coor(2,i)
         if ( indprh .eq. 0 ) then
            value = usol(i)
         else
            value = usol ( kprobh(i) )
         end if
         write ( *, 50 ) i, x, y, value
 50      format ( ' node ', i3, ' x,y ', 2d12.3, ' value ', d12.3 )
100   continue
      end

      subroutine prin2 ( npoint, coor, usol, kprobp )
      implicit none
      integer npoint, kprobp(npoint,3)
      double precision coor(2,npoint), usol(3*npoint), x, y, value

c     --- a two-dimensional mesh is assumed
```

```
c         Print USOL

      do 100 i = 1, npoint
         x = coor(1,i)
         y = coor(2,i)
         value = usol ( kprobp(i,2) )
         write ( *, 50 ) i, x, y, value
 50      format ( ' node ', i3, ' x,y ', 2d12.3, ' value ', d12.3 )
100   continue
      end

      subroutine prin3 ( npoint, coor, usol, kprobp )
      implicit none
      integer npoint, kprobp(npoint,*)
      double precision coor(2,npoint), usol(*), x, y, value

c     --- a two-dimensional mesh is assumed
c         Print USOL

      do 100 i = 1, npoint
         x = coor(1,i)
         y = coor(2,i)
         if ( kprobp(i,3) .gt. 0 ) then
            value = usol ( kprobp(i,3) )
            write ( *, 50 ) i, x, y, value
 50         format ( ' node ', i3, ' x,y ', 2d12.3, ' value ', d12.3 )
         end if
100   continue
      end

      subroutine manipulate ( npoint, coor, vector, own, work )
      implicit none
      integer npoint
      double precision coor(2,npoint), vector(npoint), own(npoint),
     +                 work(npoint)

c     --- a two-dimensional mesh is assumed
c         Print USOL

      write ( *, 100 ) (i, coor(1,i), coor(2,i), vector(i),i=1,npoint )
100   format ( ' nodes   x-coor      y-coor     value ', (i3,3d12.3) )
            .
            .
            .
            .
            +      Statements to fill array OWN
            .
            +      Use the work space in array WORK
            .
            .
            .
      end
```

22.8 Detailed description of the extra memory management tools

22.8.1 Subroutine INIACTMK

Description

General memory management subroutine that is used to activate a permanent array. If this array does not exist the new array is created if possible. If it is not possible to create the new array an error message is given.
Activation is exactly the same as in INI070 (22.6.1).
Creation of the array is only possible for a limited number of cases.

Heading

```
subroutine INIACTMK ( ibuffr, ichoice, ipos, intarr, intarrex, mmindex )
```

Parameters

INTEGER IBUFFR(∗), ICHOICE, IPOS, INTARR(∗), INTARREX(∗), MMINDEX

IBUFFR General integer buffer array (as stored in blank common), that contains all large data in SEPRAN.

ICHOICE Choice parameter indicating which array must be activated or created.
Possible values:

1 A standard subarray of KMESH.
2 A subarray of KMESH part aa.
3 A subarray of KMESH part ab.
4 A subarray of KMESH part y.
5 A subarray of KMESH part ac.

IPOS Indicates which position in the main array defines the array to be activated or created.

INTARR Contains the main array from which a part must be created or activated. Which array it concerns depends on the parameter ICHOICE.
In case $1 \leq$ ICHOICE ≤ 3 INTARR must be array KMESH.

INTARREX An extra array that may be necessary to create the subarray.
At this moment this array is not yet used.

MMINDEX Contains at output the memory management sequence number of the array that is activated or created.
MMINDEX may be used to compute starting addresses in IBUFFR or BUFFER by: IPadres = iniget(mmindex) or IPadres = inidgt(mmindex).

Input

ICHOICE and IPOS must have a value.
Array INTARR must have been filled.
Depending on ICHOICE also array INTARREX must have been filled.

Output

MMINDEX has got a value.
It is guaranteed that the array corresponding to the sequence number MMINDEX is positioned in IBUFFR.
If the position in INTARR referring to MMINDEX was zero it has got the value MMINDEX.

23 Error messages and subroutines

23.1 Introduction

In this chapter it is described how the error messages in SEPRAN are treated.
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, SEPRAN 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 23.2. Besides that, all error messages should be available in a standard ASCII file. This ASCII file is read by the main program makef4 and its contents is put into the direct access error message file. See 23.3 for a description of how to use makef4. Section 23.3 also contains a description of the ASCII file. Besides that, 23.3 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.

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 is provided by a trace back containing a list of calling subroutines.
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 23.4.
For the actual printing of error messages in a subroutine, the subroutines errsub (23.5.1) (errors) and errwar (23.5.2) (warnings) must be used, as described in 23.5.

23.2 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 CMCDPI (21.5). 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 (21.17).
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 23.3 are also stored in the error message file. They are replaced by integers, reals and characters by subroutine errsub or errwar as described in 23.5.

23.3 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 makef4 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 23.2. 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 errsub (23.5.1) and errwar (23.5.2) (23.5).

An extra option is the use of the tilde character ($\tilde{}$) followed by one of the $, & or % signs. If tilde followed by such a sign is found all remaining integers, reals or strings are printed in the form of a list. (8 integers per line, 5 reals or 1 string).

Program makef4 reads the ASCII file and creates or overwrites the direct access error message file. All error messages must be provided on one ASCII file.
Users may supply their own error messages provided they follow the rules given in Section 23.4.

Example of an ASCII error message file:

```
1:
 ichois has wrong value ($)
2:
 length of array % is too small.
 The computed length of the array is $, the declared length as stored in
 position 1 of the array is $.
 Either the array is too short, or you did not make the first position equal
 to the declared length.
3:
```

```
 Element has $ nodal points, whereas for this type number $ nodes are
 required. Element number: $, type number $, element group $
4:
 Number of element groups read is $, which is less than one or more than $
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 makef4 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 makef4 for this possibility.

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

As described earlier (see 23.2 and 23.3) 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    -  899:  Standard SEPRAN errors
900  - 1099:  Special SEPRAN errors for mesh generator by Niek Praagman
1100 - 1999:  Standard SEPRAN errors
2000 - 2099:  Errors in VIP
2100 - 2199:  Errors in spectral elements
2200 - 2299:  Errors in VISCEL
2300 - 2999:  Standard SEPRAN errors
3000 - 3499:  User provided errors
```

So the user is free to supply his own error messages in the range 3000 to 3499. The best way to handle is to keep a separate file with these error messages and to update the error message file with this separate file.
Mark that only the system administrator is able to change the error message file.
At each update of SEPRAN the error message file is renewed so if a user provides his own error messages, he must update the error message file each time a new SEPRAN version is delivered.

23.5 Available subroutines with respect to error messages and warnings

At this moment SEPRAN contains two subroutines that may be utilized for printing errors and warnings. Error messages are printed by errsub and warnings by errwar.
Since an important part of the information is given in the commons cmessc and cmessg, also subroutines are available to fill these commons. In fact there is no need to store the commons cmessc (23.5.3) and cmessg (23.5.3) in your subroutines. The subroutines errint (23.5.4), erreal (23.5.5) and errchr (23.5.6) store integers, reals and strings respectively into the common blocks.
To create the name of the calling subroutine in a tree structure the subroutines eropen (23.5.7) and erclos (23.5.8) must be used.

A typical SEPRAN subroutine has the following shape:

```
      subroutine .......

          .
          .
          .        Standard declarations etcetera
          .

      call eropen ( 'name of subroutine' )

          .
          .
          .    statements
          .
          .

    Error messages in the following sense:

      call errint ( ..., 1 )
      call errint ( ..., 2 )
          .
          .
      call errchr ( ... , ... )
      call errsub ( num, nint, nreal, nchar )
          .
          .
          .

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

The subroutines eropen and erclos must be called exactly once in each subroutine.
For main SEPRAN subroutines, in which also the time is printed and the priorities are decreased, erclos may be replaced by erclmn.
The subroutines errsub and errwar may be used as described in the next sections.

23.5.1 Subroutine errsub

Description

errsub prints an error message.
errsub does not actually stop the program except when more than maxerr errors occur. The programmer may use common block CCONST (21.3) 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 SEPRAN files are properly closed.

Heading

```
subroutine errsub ( 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 cmessc and cmessg. These common blocks may be filled directly, but it is preferred to use the subroutines errint, erreal and errchr.

Output

Print of the error message.

23.5.2 Subroutine errwar

Description

errwar prints a warning.
The printing of warnings by subroutine errwar is automatically suppressed after maxwar warnings. The programmer may use common block CCONST (21.3) to find out whether warnings have been printed and how many. errwar uses the same error message file as errsub. So an error in the file printed by errsub, may turn into a warning if printed by errwar.

Heading

```
subroutine errwar ( 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 cmessc and cmessg. These common blocks may be filled directly, but it is preferred to use the subroutines errint, erreal and errchr.

Output

Print of the warning.

23.5.3 The common blocks cmessc and cmessg

The common blocks cmessc and cmessg are defined as follows:

```
      character * 80 chars
      character * 8 namsub
      common /cmessc/ chars(1000), namsub(20)
      save /cmessc/
c
c                /cmessc/
c    Contains characters for the error message subroutines
c    Must be used in co-operation with common block cmessg
c
c    chars     Array containing character information for error messages
c              At most 1000 positions are available
c    namsub    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              levsub (see cmessg) the actual subroutine (i.e. the bottom
c              of the tree)
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      integer ints, levsub
      double precision reals
      common /cmessg/ reals(1000), ints(1000), levsub
      save /cmessg/
c
c                /cmessg/
c    Contains extra information for the error message subroutines
c    Must be used in co-operation with common block cmessc
c
c    ints      Array containing integer information for error messages
c    levsub    Level of present subroutine for traceback
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 eropen, erclos, errint, erreal and errchr to fill these commons.

23.5.4 Subroutine errint

Description

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

Heading

```
subroutine errint ( 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 cmessc

23.5.5 Subroutine erreal

Description

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

Heading

```
subroutine erreal ( 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 cmessc

23.5.6 Subroutine errchr

Description

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

Heading

```
subroutine errchr ( 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 cmessg.

23.5.7 Subroutine eropen

Description

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

A call of eropen must always be followed by a call to erclos or erclmn in the same subroutine!

Heading

```
subroutine eropen ( 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 levsub has been raised.

23.5.8 Subroutine erclos

Description

SEPRAN subroutine, that removes the name of the local subroutine from the parameter name in common cmessg/cmessc by reducing levsub

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

Heading

```
subroutine erclos ( namelc )
```

Parameters

character *(*) namelc

namelc Name of local subroutine that must be removed

Input

namelc must have a value.

Output

The parameter levsub has been decreased.

23.5.9 Subroutine erclmn

Description

Subroutine erclmn does the final closing of a main SEPRAN subroutine. It is supposed that the subroutine has been opened with eropen and that erclos has not been called erclmn performs the following tasks:
call the timing subroutine depending on the parameter itime.
decrease all priorities of the arrays in ibuffr (22.5).
call erclos, i.e. release the name of the main subroutine from name.

Heading

```
subroutine erclmn( namenw )
```

Parameters

character *(*) namenw

namenw Name of calling subroutine

Input

namelc must have a value

Output

The parameter levsub has been decreased.

24 The SEPRAN arrays

24.1 Introduction

In this section we shall describe the structure of all standard SEPRAN arrays. These arrays consist of some some information concerning the structure of the arrays itself as well of one or more sequence numbers referring to arrays stored in the large buffer. The arrays stored in the buffer array can only be approached by the memory management subroutines described in Section 22.

In this chapter we describe the following arrays:

24.2 Array KMESH contains information about the mesh used.

24.3 Array KPROB contains information about the problem definition.

24.4 Array ISOL contains information about the solution vector.

24.5 Array IRHSD contains information about the right-hand side vector.

24.6 Array INTMAT contains information about the structure of the large matrix as stored in array MATR.

24.7 Array MATR contains information about the large matrix. This information should always be used in combination with INTMAT.

24.8 Array IVECTR contains information about vectors that have a structure that is different from the solution vector.

24.9 Array IMAP Contains integer information concerning the mapping of one mesh to another one.

Each SEPRAN array gets a specific type number. This type number is uses as identification of the type of array. The following type numbers are presently in use:

100 Array KMESH (24.2)

101 Array KPROB (24.3)

102 Array INTMAT (24.6)

103 Array MATR (24.7)

110 Solution array USOL (24.4) or right-hand-side vector IRHSD (24.5)

115 Array of special structure (24.8)

116 Array of special structure defined per element (24.8)

117 User array containing function values or co-ordinates (18.3.1)

118 Array containing information concerning a particle trace (24.8)

119 Special array to store information on curves (24.8)

124 Array of special structure corresponding to edges per node

125 Array of special structure corresponding to faces per node

126 Array of special structure corresponding to nodes per element

127 Array of special structure corresponding to edges

128 User array containing node numbers (18.3.3)

130 Array MAP (24.9)

24.2 Array KMESH

Array KMESH contains all information about the mesh, such as the topological description, information about neighbors and so on. A small part of the information is stored directly in array KMESH. This concerns some constants and some small subarrays. Most of the information is stored in the buffer array IBUFFR and can only be reached with the aid of the memory management subroutines. For those arrays the so-called sequence numbers are stored in KMESH.
KMESH consists of two parts.

Part 1: (pos. 1-80) contains special variables and pointers.

Part 2: (pos. 81 - end) contains the sub arrays not stored in IBUFFR.

Part 1 This part consists of exactly 80 positions. These positions have the following meaning:

Pos. 1 declared length of array KMESH (to be filled by the user)

Pos. 2 100. This number is used to indicate that this is an array of structure KMESH.

Pos. 3 actual length of array KMESH.

Pos. 4 NPELM: maximal number of points in element

Pos. 5 NELGRP: number of element groups as defined in the mesh generation

Pos. 6 NDIM: dimension of the space.

Pos. 7 NLEVEL: number of levels, see KMESH part k.

Pos. 8 NPOINT: number of nodal points.

Pos. 9 NELEM: number of elements.

Pos. 10 NUSPNT: number of user points.

Pos. 11 NCURVS: number of curves.

Pos. 12 NSURFS: number of surfaces.

Pos. 13 NVOLMS: number of volumes.

Pos. 14 KELGRP: $\sum_{IELGRP=1}^{NELGRP}$ number of points in standard element IELGRP

Pos. 15 KELMA: Starting address of KMESH part a in KMESH.
Part a corresponds to the number of points in the elements.

Pos. 16 KELMB: Starting address of KMESH part b in KMESH.
Part b corresponds to the boundary nodal points in standard elements.

Pos. 17 KELMC: Memory management sequence number of KMESH part c.
Part c contains the nodal point numbers of the elements sequentially.

Pos. 18 KELMD: Memory management sequence number of KMESH part d.
Part d contains the elements corresponding to the nodal points.

Pos. 19 KELME: Memory management sequence number of KMESH part e.
Part e contains a list of the nodal points connected to the nodal points.

Pos. 20 KELMF: Starting address of KMESH part f in KMESH.
Part f contains the shape numbers of the elements

Pos. 21 KELMG: Starting address of KMESH part g in KMESH.
Part g contains the number of elements for the different element groups.

Pos. 22 KELMH: Memory management sequence number of KMESH part h.
Part h contains information of the points, curves, and surfaces.

Pos. 23 KELMI: Memory management sequence number of KMESH part i.
Part i contains the co-ordinates.

Pos. 24 KELMJ: Memory management sequence number of KMESH part j.
Part j contains the new numbering of the nodal points when renumbering is performed.

Pos. 25 KELMK: Memory management sequence number of KMESH part k.
Part k contains the levels.

Pos. 26 KELML: Memory management sequence number of KMESH part l.
Part l contains the nodal point numbers of the user points.

Pos. 27 KELMM: Memory management sequence number of KMESH part m.
Part m contains the nodal point numbers of the curves.

Pos. 28 KELMN: Memory management sequence number of KMESH part n.
Part n contains the nodal point numbers of the surfaces.

Pos. 29 KELMO: Memory management sequence number of KMESH part o.
Part o contains the neighbor elements of each element.

Pos. 30 KELMP: Memory management sequence number of KMESH part p.
Part p contains the length of the begin and end elements for all curves.

Pos. 31 KELMQ: Memory management sequence number of KMESH part q.
Part q contains subcurve numbers of each composite curve.

Pos. 32 KELMR: Memory management sequence number of KMESH part r.
Part r contains subsurface numbers of each composite surface.

Pos. 33 KELMS: Memory management sequence number of KMESH part s.
Part s contains the curve numbers corresponding to each subsurface.

Pos. 34 IINPUT: Memory management sequence number of array IINPUT.
This part contains the array IINPUT as described in Section 3.3.
This array is always created internally if subroutine MESH (3.2) is called.

Pos. 35 RINPUT: Memory management sequence number of array RINPUT.
This array contains the array RINPUT as described in Section 3.3
This array is always created internally if subroutine MESH (3.2) is called.

Pos. 36 KELMT: Starting address of KMESH part t in KMESH.
Part t contains the element numbers of the elements with highest dimensionality.

Pos. 37 KELMZ: Starting address of KMESH part z in KMESH.
Part z contains for each curve an indication whether it has been changed or not.

Pos. 38 KELMU: Indicates whether spectral elements will be used or not. If KELMU=0 no spectral elements are available, if KELMU>0, it indicates the starting address of KMESH part u in array KMESH.

Pos. 39 reserved

Pos. 40 NMESH:Number of various mesh stored in KMESH (not yet in use).

Pos. 41 NEXTPNT: Number of extra points generated in case the user provided the nodes and SEPRAN generated a corresponding mesh.

Pos. 42 NUMOBST: Number of obstacles in the mesh.

Pos. 43 ICHANGE_OBST: This parameter has a special meaning. If 0 it means that in the new mesh no obstacle points are activated or deactivated, hence the type of boundary conditions do not change. If 1 it means that new nodes are activated or deactivated.

Pos. 44 NSPLIT: Number of disjoint parts the mesh consists of. If the mesh consists of 1 part only, NSPLIT=0.

Pos. 45 KELMV: Memory management sequence number of KMESH part v.
Part v contains information about element groups coupled to element groups.

Pos. 46 KELMW: Starting address in array KMESH of information concerning the real and integer properties of element groups.

Pos. 47 KELMX: Memory management sequence number of KMESH part x.
Part x contains information with respect to obstacles.

Pos. 48 KELMAA: Starting address in array KMESH part aa. This part is very special, since it contains all kinds of information about faces and edges. Also some of the information stored in previous parts can be found in this part, however, using an alternative storage.

Pos. 49 KELMAB: Starting address in array KMESH part ab. This part contains information about curves, surfaces and volumes, organized in a way such that it can be information can be found in a very simple way.

Pos. 50 KELMY: Starting address in array KMESH concerning KMESH part y. Part y contains information needed for the unstructured finite volume flow solver, for which a SEPRAN mesh is required.

Pos. 51 KELMAC: Starting address in array KMESH concerning KMESH ac. Part ac contains information of the position of all nodal points in a rectangular grid. This simplifies the search for points in the neighborhood of a given coordinate.

Pos. 52 KELMAD: Starting address in array KMESH concerning KMESH ad. Part ad contains information of the obstacles in relation to the mesh

Pos. 53 Memory management sequence number of array transportinfo.
Is only used for parallel computing

Pos. 54 Memory management sequence number of array blockinfo.
Is only used for parallel computing

Pos. 55 KELMAE: Memory management sequence number of array `IELGRP_NODES` containing the nodes corresponding to the element groups

Pos. 56 KELMAF: Memory management sequence number of array `NODE_ELGRPS` containing the element groups corresponding to the nodes

Pos. 57 KELMAG: Memory management sequence number of array `VOLUME_SURFS` containing the surface numbers corresponding to the volumes

Pos. 58 KELMAH: Memory management sequence number of array iouterboun containing the outer curves or surfaces provided with a sign, such that the normal on these boundaries is always directed outwards.

Pos. 59 This position is reserved for use in local arrays. Sometimes an array is created using this position as memory management sequence number and removed later on.

Pos. 60 KELMAI: Memory management sequence number of array surface_nodes containing the number of nodes as well as the node numbers of all outer surfaces.

Pos. 61 KELMAJ: Memory management sequence number of array normal containing the normals on the outer surfaces.

Pos. 62 KELMAK: Memory management sequence number of array `CURVE_USERP` containing the user point numbers corresponding to the curves.

Pos. 63 KELMAL: Memory management sequence number of array KMESH part al, containing the outer boundary of sets of element groups.

Pos. 64 NLEVELSETS (number of different level set functions that are used to update the mesh)

Pos. 65 KMESHAM Starting address of array KMESHPAR in KMESH. This array contains some information about the global mesh in case of parallel computing.

Pos. 66-80 Not yet used

Part 2 In this part we describe both the subarrays of KMESH that are stored in KMESH as well as the arrays stored in the buffer array IBUFFR to which KMESH points.

part a corresponds to the number of points in the elements. It is only used when KELMA > 0. This implies that the number of nodes is not the same for all standard elements.

Length: NELGRP + 1

First position in KMESH: KELMA

Contents: number of nodal points in each standard element accumulated.
Hence: KMESH(KELMA) = 0
Let INPELM be the number of nodal points in standard element i. Then
INPELM = KMESH(KELMA+i) - KMESH(KELMA+i-1)

part b corresponds to the boundary nodal points in standard elements. This part is only used when KELMB > 0.

Length: NELGRP

First position in KMESH: KELMB

Contents: For each standard element the number of boundary nodal points is stored.
Hence: KMESH(KELMB+IELGRP-1) = number of boundary nodal points in standard element IELGRP.

part c contains the nodal point numbers of the elements sequentially.
This part is always used and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELMC as sequence number.
Length: $\sum_{IELEM=1}^{NELEM}$ number of nodal points in element IELEM

Contents: nodal point numbers of each element sequentially. The elements must be numbered according to increasing standard element number. Hence first all elements with standard element 1, then with standard element number 2, etc.

Example

Consider the mesh in Figure 24.2.1, with two standard elements (triangle and quadrilateral)

KMESH part *c* becomes:

```
1, 3, 2, 1, 4, 3, 3, 6, 5, 2, 6, 9, 8, 5, 4, 7, 6, 3, 7, 10, 9, 6
```

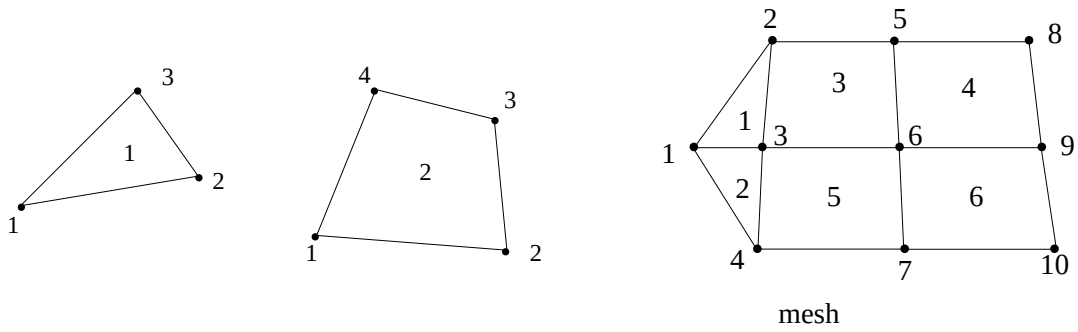


Figure 24.2.1: Standard elements

part d contains the elements corresponding to the nodal points.
This part is used if KELMD>0 and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELMD as sequence number.
Length:

NPOINT+1+ $\sum_{INODP=1}^{NPOINT}$ number of elements corresponding to nodal point INODP

Contents: Part d contains for each nodal point the elements it is lying in.
It consists of two parts:

The first part ($d1$) contains the number of elements nodal point INODP is lying in, accumulated in the following way:

KMESH $d1$ (1) = 0

number of elements nodal point INODP is lying in =
KMESH $d1$ (INODP+1) - KMESH $d1$ (INODP)

The second part ($d2$) contains the actual element numbers for the nodal points sequentially. Hence the first element in which nodal point INODP lies is given by:

KMESH $d2$ (KMESH $d1$ (INODP) + 1)

Example

Consider the mesh in Figure 24.2.2

KMESH part $d1$ becomes:

```
0, 2, 4, 8, 10, 12, 16, 18, 19, 21, 22
```

KMESH part $d2$ becomes:

```
1, 2, 1, 3, 1, 2, 3, 5, 2, 5, 3, 4, 3, 4, 5, 6, 5, 6, 4, 4, 6, 6
```

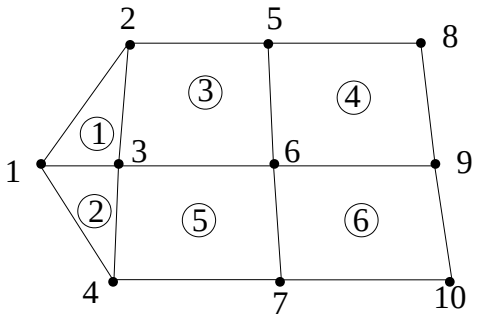


Figure 24.2.2: mesh

part e contains a list of the nodal points connected to the nodal points.
This part is used if KELME>0 and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELME as sequence number.
Length:
NPOINT+1+ $\sum_{INODP=1}^{NPOINT}$ number of nodal points connected with nodal point INODP

Contents: Part e contains for each nodal point the nodal points it is connected with.
It consists of two parts:

The first part ($e1$) contains the number of nodal points, nodal point INODP is connected with, accumulated in the following way:

KMESH $e1$ (1) = 0

number of nodal points, nodal point INODP is connected with =
KMESH $e1$ (INODP+1) - KMESH $e1$ (INODP)

The second part ($e2$) contains the actual nodal point numbers sequentially. Hence the first nodal point, INODP is connected with is given by:

KMESH $e2$ (KMESH $e1$ (INODP) + 1)

Example

Consider the mesh in Figure 24.2.2 KMESH part $e1$ becomes:

```
0, 3, 7, 13, 17, 22, 30, 35, 38, 41, 44
```

KMESH part $e2$ becomes:

```
4, 2, 3, 1, 6, 3, 5, 1, 4, 7, 6, 5, 2, 1, 3, 7, 6, 2,
3, 6, 9, 8, 2, 3, 4, 7, 10, 9, 8, 5, 3, 4, 10, 9, 6
```

part f contains the shape numbers of the elements. This subarray is always stored in KMESH.
Length: NELGRP positions

First position in KMESH: KELMF

Contents: This part contains for every standard element a type number (MESHTP) indicating the type of element it is for the mesh generators.
Possibilities of type number MESHTP:

$<$ -10: line element with - MESHTP - 9 nodal points.
-9: line element containing all points on the side.
1: line element with 2 points.
2: line element with 3 points.
3: triangle with 3 points.
4: triangle with 6 points.
5: quadrilateral with 4 points.
6: quadrilateral with 9 points.
7: triangle with 7 points.
8: triangle with 10 points.
9: quadrilateral with 5 points.
11: tetrahedron with 4 points.
12: tetrahedron with 10 points.
13: hexahedron with 8 points.
14: hexahedron with 27 points.
15: tetrahedron with 14 points.
16: tetrahedron with 15 points.

part g contains the number of elements for the different element groups. This part is only used when NELGRP $>$ 1.

Length: NELGRP positions

First position in KMESH: KELMG

Contents: The number of elements of standard element IELGRP is stored in KMESH(KELMG+IELGRP-1).

part h contains information of the points, curves, and surfaces. This part is used if KELMH>0 and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELMH as sequence number.
Length: 2 × (3 + (NSURFS + NVOLMS) × 3)
KMESH part h has only been filled if subroutine MESH has been called explicitly.

Contents: Part h contains the memory management sequence numbers and length of the arrays describing the user points, curves, surfaces and volumes, in the following way:

Pos. 1: Memory management reference number of array ICURVSOBST. This array has length 2 × ICURVS(NCURVS), see position 3. If ICURVSOBST$(1, i)$ = IOBST then the corresponding nodal point is positioned at an obstacle with sequence number IOBST. Otherwise (0) the node is free.
ICURVSOBST$(2, i)$ contains the relative position in the obstacle curve.

Pos. 2: Memory management reference number referring to array IOBSTACT. This array has length NUMOBST. If there are possibly nodal points at obstacle IOBST then IOBSTACT(IOBST) = 1, otherwise IOBSTACT(IOBST) = 0.

Pos. 3: Memory management reference number of array ICURVS.
ICURVS consists of two parts each of length NCURVS.
Part 1 contains the number of nodal points on the curves accumulated. This means that in position ICURVS(1) the number of nodes in curve 1 can be found. The number of nodes in curve $i, i > 1$ is given by ICURVS(i)-ICURVS($i-1$).
The second part of icurvs indicates if curve i is a standard curve ICURVS(NCURVS+i) = 0, or an empty or composite curve (ICURVS(NCURVS+i) = 1).

Pos. 4: Memory management reference number of array IOBSCURV. This array has length NCURVS.
It indicates if a curve is on an obstacle or not.
iobscurv(i) may take one of the following values:

0 means curve i not on an obstacle.

1 points of curve i on an obstacle.

2 only begin and end point on obstacle.

3 only begin point on obstacle.

4 only end point on obstacle.

Pos. 5: Memory management reference number of array containing the co-ordinates of the curves.

Pos. 6: Memory management reference number of array containing the factors t defined along the curves.

The next 6 × (NSURFS + NVOLMS) positions contain the information of the surfaces and volumes in that sequence. For each surface or volume, six positions are required:

Pos. (i) : contains the memory management reference number of the array, containing the co-ordinates of the surface or volume.
The storage of this array is the same as array COOR.

Pos. (ii) : contains the memory management reference number of the array, containing the elements of the surface or volume.
The storage of this array is the same as array KMESH part c.

Pos. (iii) : contains the memory management reference number of the array, containing the global numbers of the local nodal points of the surface or volume.

Pos. (iv) : contains the memory management reference number of the array, containing the array KBNDPT, describing the boundary of the surface or volume.
In R^2 this is an array containing the nodes of the boundary sequentially, in R^3 this array contains the node numbers of the elements of the surface.

Pos. (v) : contains the memory management reference number of the array, containing the renumbering array RENUM to be used by the surface generator GENERAL. In other cases this array is empty.

Pos. (vi) : May be changed in the future.
At this moment the number of elements in the surface or volume is stored.

The numbering of the nodal points is a local one, furthermore the same storage as for the global elements is used.

part i contains the co-ordinates.
This part is always used and is stored in the buffer array BUFFER. Its starting address can be found with the memory management subroutines using KELMI as sequence number.
Mark that this is a double precision (real) array in contrast with almost other parts of KMESH which are integer. Length: NPOINT × NDIM real positions.

Contents: co-ordinates of the nodal points sequentially. Hence first all co-ordinates of the first point, then of the second one, etc.

Locally KMESH part i may be treated as a double precision two dimensional array COOR of size: COOR(1:NDIM,1:NPOINT).

part j contains the new numbering of the nodal points when renumbering is performed.
This part is used if KELMJ>0 and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELMJ as sequence number.
Length: NPOINT integer positions.

Contents: the new numbering of the nodal points, when renumbering is performed. Hence the first node to be used is KMESHJ(1), the second one KMESHJ(2) and so on.

part k contains the levels.
This part is used if KELMK>0 and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELMK as sequence number.
Length: NLEVEL + NSPLIT integer positions.

Contents: First the NLEVEL levels in the mesh are stored. NLEVEL is computed by the program. The description of the levels can be found in the Users Manual Section 3.2.2 in the part RENUMBER.
The end points of the levels are stored. Hence if for example KMESH k contains: 4, 8, 12, 16 then indicating that level 1 contains the nodal points 1 to 4, level 2 the points 5 to 8, etc.
In this case nodal points 1 to 4 means nodes 1 to 4 if KELMJ=0 and KMESHJ(1), KMESHJ(2), KMESHJ(3), KMESHJ(4) if KELMJ>0.

The positions NLEVEL+1 to NLEVEL+NSPLIT contains information about the parts. KMESHk(NLEVEL+i) contains the highest nodal point sequence number in part i

part l contains the nodal point numbers of the user points.
This part is always used and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELML as sequence number.
Length: NUSPNT positions.

Contents: The nodal point numbers corresponding to the user points are stored sequentially. Hence the nodal point number corresponding to user point i is stored in KMESH l (i).
When KMESH l (i) = 0, then point i is not a nodal point.

part m contains the nodal point numbers of the curves.
This part is always used provided NSUREL > 0 or NVOLEL > 0 and is stored in the

buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELMM as sequence number.
Length: 5 + NINNER + NCURVS + number of nodal points in the curves.

Contents: The nodal point numbers of the curves are stored in this part.
Part m consists of four subparts:

part (i): length: 4 positions, contains some parameters

pos. 1 NINNER: number of inner boundaries

pos. 2 starting position of part (iii)

pos. 3 starting position of part (iv)

pos. 4 length of part (iv)

part (ii): length: 1 + NCURVS positions, contains the number of nodal points in the curves accumulated. The last position contains the number of nodal points plus one, the first position 1.

part (iii): length: LENGM3 = number of nodal points in the curves, contains the nodal point numbers of the curves sequentially.

part (iv): length: NINNER positions, contains the curve number of the inner boundaries sequentially in increasing curve number.

Example

Consider the boundaries in Figure 24.2.3.

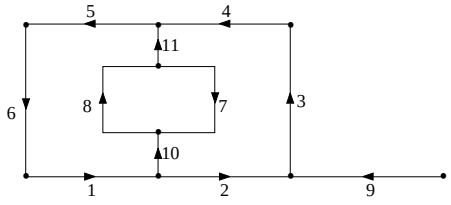


Figure 24.2.3: 11 curves (2 inner boundaries)

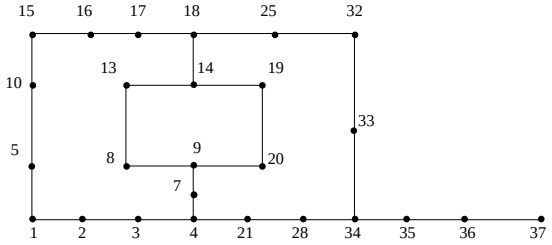


Figure 24.2.4: nodal point numbers

Let the nodal points be given in Figure 24.2.4. Then KMESH part m becomes:

```
Part (i):    2, 17, 56, 2
Part (ii):   1, 5, 9, 12, 15, 19, 23, 27, 31, 35, 38, 40
```

```
Part (iii):  1, 2, 3, 4, 4, 21, 28, 34, 34, 33, 32, 32, 25, 18, 18, 17,
             16, 15, 15, 10, 5,  1, 14, 19, 20,  9,  9,  8, 13, 14, 37,
             36, 35, 34,  4, 7,  9, 14, 18
Part (iv):   10, 11
```

part n contains the nodal point numbers of the surfaces. This part is only used when NVOLEL > 0 and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELMN as sequence number.
Length: $2 \times$ NSURFS + number of nodal points in the surfaces.

Contents: The nodal point numbers of the surfaces are stored in this part.
Part n consists of three subparts:

part (i): length: NSURFS positions, contains the number of nodal points in the elements of the surfaces sequentially.

part (ii): length: NSURFS positions, contains the number of elements in the surfaces sequentially.

part (iii): length: $\sum_{ISURF=1}^{NSURFS}$ number of elements $\times$ number of nodal points per element of surface ISURF.

In this part the nodal point numbers of the elements of the surfaces are stored sequentially. Hence first all nodes of element 1 of surface 1, then all nodes of element 2 of surface 1 etc.

part o contains the neighbor elements of each element.
This part is used if KELMO>0 and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELMO as sequence number.
Length:

NELGRP + $\sum_{ielgrp=1}^{nelgrp}$ Number of neighbors of each element in element group ielgrp

Contents: Part o consists of two subparts $o1$ and $o2$:

part (i): length: NELGRP positions, contains the number of boundaries of each element in the standard element group IELGRP. So KMESH O1 (ielgrp) contains the number of boundaries of elements with element group number ielgrp.

For triangular elements the number of boundaries is always three, for quadrilaterals four, for tetrahedrons it is four and for brick type elements it is six. At this moment other types of elements are not filled in array KMESH part o.

part (ii): length: $\sum_{ielgrp=1}^{nelgrp} numelements_{ielgrp} \times KMESHO1(ielgrp)$ positions, contains the actual information.

Here $numelements_{ielgrp}$ is the number of elements with element group number ielgrp.

For each element KMESH O1(ielgrp) positions are used (elements are stored sequentially).
For each boundary of the element (i.e. lines in two-dimensional elements, and planes in three-dimensional ones), the element connected to that same boundary is stored. For 2D elements not only the element number but also the boundary number of that neighboring element is stored.

For 2D elements the first boundary is defined as the line from first vertex to second vertex (as stored in KMESH part c), the second boundary as the line from second

vertex to third vertex etc.

Let k be the position in KMESH part O2 corresponding to the j^{th} boundary of the i^{th} element. Then:

KMESH O2(k) = 0 means that this boundary does not have another element connected to it. Hence it must be a part of the outer boundary.

KMESH O2(k) > 0 indicates that boundary j of element i is connected to an element.
If all elements are two-dimensional (i.e. triangles or quadrilaterals) then KMESH O2(k) is equal to IELEM + (NELEM+1) × KBOUNDARY, where IELEM denotes the element number of the element connected to element i, NELEM denotes the total number of elements and KBOUNDARY denotes the boundary number of the boundary in IELEM that is connected to element i.
If at least one element is three-dimensional (tetrahedrons or bricks) then KMESH O2(k) is equal to IELEM only.

part p contains the length of all begin and end elements of all curves. This part is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELMP as sequence number.
Length: 2 × NCURVS reals.

Contents: Part p may be considered as a real array elsize of size (2,NCURVS).
elsize(1,i) contains the length of the first element in curve i.
elsize(2,i) contains the length of the last element in curve i.
Array elsize is only filled the first time a mesh is created. If the mesh is adapted elsize is kept, so that it can be used to find the original element sizes.

part q contains subcurve numbers of each composite curve. This part is used if KELMQ>0 (i.e. if curves of curves are defined) and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELMQ as sequence number.
Length: NCURVS + 1 + $\sum_{icurve=1}^{NCURVS}$ Number of subcurves of curve $icurve$

Contents: Part q consists of two subparts $q1$ and $q2$:

part (i): length: NCURVS+1 positions, contains the number of subcurves each curve contains (accumulated) in the following way
KMESH Q(k+1) - KMESH Q(k) = number of subcurves of curve k
KMESH Q(1) contains the last position of part (i), hence NCURVS+1.

part (ii): length: $\sum_{icurve=1}^{NCURVS}$ Number of subcurves of curve $icurve$ positions, contains the subcurve numbers of the respective curves in the sequence defined by part (i)

part r contains subsurface numbers of each composite surface. This part is used if KELMR>0 (i.e. if surfaces of surfaces are defined) and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELMR as sequence number.
Length: NSURFS + 1 + $\sum_{isurf=1}^{NSURFS}$ Number of subsurfaces of surface $isurf$

Contents: Part r consists of three subparts $r1$, $r2$ and $r3$:

part (i): length: NSURFS+1 positions, contains the number of subsurfaces each surface consists of (accumulated) in the following way:
KMESH r(k+1) - KMESH r(k) = number of subsurfaces of surface k
KMESH r(1) contains the last position of part (i), hence NSURFS+1.

part (ii): length: $\sum_{isurf=1}^{NSURFS}$ Number of subsurfaces of surface $isurf$ positions.
Contains the subsurface numbers of the respective surfaces in the sequence defined by part (i)

part (iii): Contains for each subsurface in the composite surface an index number, referring to the first record number from which array IPOINT is stored in file 3.

Array IPOINT of length NPOINT+NELEM contains the following information:

Pos. 1 - NPOINT contains the mapping from local numbering in subsurface to global numbering in surface. (NPOINT denotes the number of nodes in the subsurface).

Pos. NPOINT+1 - NPOINT+NELEM contains the mapping from local element numbering in subsurface to global numbering in surface. (NELEM denotes the number of elements in the subsurface).

part s contains the curve numbers corresponding to each subsurface. This part is used if KELMS>0 (i.e. if NSURFS $>$ 0) and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELMS as sequence number.
Length: NSURFS+1+8×NSURFS+$\sum_{isurf=1}^{NSURFS}$ Number of curves surrounding surface $isurf$

Contents: Part s consists of three subparts $s1$, $s2$ and $s3$:

part (i): length: NSURFS+1 positions, contains the number of curves surrounding each surface sequentially in the following way:
KMESH s(k+1) - KMESH s(k) = number of curves in surface k
KMESH s(1) contains the last position of part (ii), hence
NSURFS+1+8×NSURFS.

part (ii): length: 8 × NSURFS positions.

Contains for each surface 8 positions with the following information:

Pos. 1 ISURF (type of surface, i.e. by which generator created, see mesh generator)
Possible values:

1. RECTANGLE
2. GENERAL
3. TRIANGLE
4. QUADRILATERAL
5. MESHUS
6. INPUT
7. SURFACES OF SURFACES
8. TRANSLATE
9. ROTATE
10. PIPESURFACE
11. ORDERED SURFACES
12. ORDERED PIPE SURFACES
13. COONS
14. SIMILAR
15. PARSURF
16. REFLECT
17. RECTANGLE with smoothing
18. ISOPAR
19. PAVER
20. SPHERE

Pos. 2 ISHAPE (shape of the elements, see mesh generator)

Pos. 3-8 The contents of these positions depend on the value of ISURF.
If fewer quantities are stored, the rest of the positions is undefined.

ISURF = 1,17 N, M

ISURF = 2,15,18 -

ISURF = 3,19 If the boundary contains holes the number of curves in the first closed part of the boundary.

ISURF = 5 NELEM, NPOINT

ISURF = 6 FILE-REFERENCE-NUMBER, NELEM, NPOINT

ISURF = 7 Number of external boundaries, number of internal boundaries, number of external boundaries that are connected
If the last number is smaller than the first one, it means that the outer boundary consists of subparts and the surface contains at least one hole.

ISURF = 8,9,14,16 surface number of surface to be copied, followed by surface number of originating surface (recursively)

ISURF = 10 Number of boundaries of curves C1, C2, C3 and C4, followed by N, M

ISURF = 11 Number of external boundaries, number of internal boundaries, N, M

ISURF = 12 Number of boundaries of curves C1, C2, C3 and C4, followed by N, M

ISURF = 20 2 extra positions are used:

3 New surface number of sphere as surface of surfaces. The actual surface may be considered as a copy of this surface

4 Curve number that is used as actual circle for the sphere. This number differs from the one given in the input file.

part (iii): length: $\sum_{isurf=1}^{NSURFS}$ Number of curves surrounding surface $isurf$ positions.

Contains the curve numbers of the surfaces sequentially in the sequence defined by part (i).
In the case of a surface consisting of subsurfaces, the curves are stored in the following sequence:
First the outer curves enclosing the surface in the correct sequence, followed by the inner curves in arbitrary order.

part t contains the element numbers of the elements with highest dimensionality. KMESH part t is always stored in array KMESH. KELMT contains the
starting address of part t in array KMESH.

Length: The length of KMESH part t depends on the dimensionality of the highest dimension used.
If volume elements are used (NVOLMS>0) the length is equal to 2 × NVOLMS.
If surface elements are used but no volume elements (NVOLMS=0, NSURFS>0) the length is equal to 2 × NSURFS.
If only curve elements are used (NVOLMS=0, NSURFS=0, NCURVS>0) the length is equal to 2 × NCURVS.

Contents: The contents of KMESH part t depend on the parameters NVOLMS, NSURFS and NCURVS.
If volume elements are used (NVOLMS>0) then KMESH t has the following contents:
KMESHt$(2 \times i - 1)$ contains the first element number of volume i.

KMESHt($2 \times i$) contains the last element number of volume i.
If surface elements are used but no volume elements (NVOLMS=0, NSURFS>0) then KMESH t has the following contents:
KMESHt($2 \times i - 1$) contains the first element number of surface i.
KMESHt($2 \times i$) contains the last element number of surface i.
If only curve elements are used (NVOLMS=0, NSURFS=0, NCURVS>0) then KMESH t has the following contents:
KMESHt($2 \times i - 1$) contains the first element number of curve i.
KMESHt($2 \times i$) contains the last element number of curve i.

part u contains information of the spectral elements. This part is only used if KELMU>0. KMESH part u is always stored in array KMESH. KMESHU contains the starting address of part u in KMESH.

Length: 5 positions

Contents:

Pos. 1 Number of different spectral groups in the mesh. At this moment only 1 group is allowed.

Pos. 2 Number of extra points in a side or edge of the spectral elements, the two end points excluded.

Pos. 3 Number of extra internal points in a surface or face of a spectral element if the boundary points are excluded.

Pos. 4 Number of extra internal points in a volume spectral element if the boundary points are excluded.

Pos. 5 Parameter that indicates in which way the extra points have to be positioned on the sides and as a consequence internally. Possible values:

0: subdivision is equidistant.

1: subdivision is in accordance with a Gauss-Legendre-Lobatto polynomial.

part v KMESH part v contains information about element groups coupled to element groups. It is only filled if KMESH(45)>0.
In that case KMESH part v is stored in array IBUFFR and the starting address may be computed by the memory management subroutines.
Length: nelgrp+1+ $\sum_{IELGRP=1}^{NELGRP}$ number of element groups coupled to element group IELGRP.

Contents: KMESH part v consists of two parts KMESHV1 and KMESHV2.

KMESHV1 of length NELGRP+1 contains the number of element groups coupled to each element group accumulated in the following way:
KMESHV1(1) = 0
KMESHV1(ielgrp+1) - KMESHV1(ielgrp) is number of element groups coupled to element group ielgrp.

KMESHV2 of length KMESHV1(NELGRP+1) contains the element group sequence numbers of the element groups coupled to the specific element group. The sequence is: first all element groups coupled to element group 1, then coupled to group 2 and so on. KMESHV1 defines the starting addresses of each element group.
The element groups coupled to IELGRP are stored in KMESHV2 positions KMESHV1(IELGRP)+1 to KMESHV1(IELGRP+1).

part w KMESH part w is only used if real or integer properties are given for the element groups. This is the case if KMESH(46)>0. Part w itself consists of 4 positions with the following contents:

Pos. 1 nintprop, i.e. the number of integer properties per element group.

Pos. 2 nrealprop, i.e. the number of real properties per element group.

Pos. 3 memory management address of array intprop with integer properties. If nintprop=0, this position contains a 0.

Pos. 4 memory management address of array realprop with real properties. If nrealprop=0, this position contains a 0.

Array intprop may be considered as a two-dimensional array of size nintprop × NELGRP. INTPROP(i, j) contains the i^{th} integer property in element group j.

Array realprop may be considered as a two-dimensional array of size nrealprop × NELGRP. REALPROP(i, j) contains the i^{th} real property in element group j.

part x KMESH part x is only used in case the mesh contains obstacles (NUMOBST>0). KMESH part x is stored in array IBUFFR . Its starting address can be found with the memory management subroutines using KELMX as sequence number.
KMESH part X consists of three arrays RMESHX, KMESHX and KMESHXa, where RMESHX is a double precision array and KMESHX and KMESHXa are integer arrays. These arrays are stored sequentially, first RMESHX and then KMESHX and KMESHXa.
Length:
RMESHX is LENIN × NUMOBST reals (double) long.
KMESHX is LENRL × NUMOBST integers long.
In R^2 we have LENIN=6 and LENRL=6.
In R^3 we have LENIN=12 and LENRL=9.
The length of KMESHXa depends on the mesh. At this moment it is zero in case of R^3

Contents:

KMESHX may be considered as a two dimensional array of size LENIN × NUMOBST with the following contents:
In R^2:

KMESHX(1,*iobst*) contains the type of obstacle, with *iobst* the obstacle sequence number. Possible values:

1 The obstacle consists of a curve.

KMESHX(2,*iobst*) contains the curve number of the obstacle.

KMESHX(3,*iobst*) contains nx.
In order to check whether a point is in the obstacle or not, the obstacle is covered by a rectangular grid of size nx × ny.

KMESHX(4,*iobst*) contains ny.

KMESHX(5,*iobst*) contains the memory management sequence number of array iobgrid. This array of size nx × ny × 2, contains information about the rectangular grid that covers the obstacle in the following way.
Let iobgrid be a three-dimensional array iobgrid(1:nx, 1:ny, 1:2).
The region covered is defined by xmin, xmax, ymin, ymax. The cell size is dx × dy Cell i, j refers to the co-ordinates: (xmin+(i-1)dx,ymin+(j-1)dy) × (xmin+idx,ymin+jdy).
iobgrid(i, j, k) may corresponds to:

iobgrid$(i, j, 1)$ may take one of the following values:

0 cell outside obstacle.

k>0 boundary part k crosses cell.

-1 cell completely in obstacle.

iobgrid$(i, j, 2)$ = k1 >0 if there is a second boundary part k that crosses the cell i, j.

KMESHX(6,*iobst*) contains the starting address of the obstacle in KMESHXa. If this starting address is 0, no information of this obstacle is stored in KMESHXa, which means that the obstacle itself is not part of a surface.

KMESHXa is only used if there are obstacles which are (partly) part of a surface. This means that some curves of the obstacle are also part of a surface. If this is the case the starting addresses of these obstacles can be found in KMESHX(6,*iobst*), where the first starting address is equal to 1.
For each obstacle in KMESHXa numsurfs+1 positions are used, where numsurfs is the number of surface that are surrounded by curves that are also part of the obstacle. The first position contains the value numsurfs, the next numsurfs positions the corresponding surface sequence numbers.

In R^3 the contents of KMESHX are:

KMESHX(1,*iobst*) contains the type of obstacle, with *iobst* the obstacle sequence number. Possible values:

2 The obstacle consists of a surface.

KMESHX(2,*iobst*) contains the surface number of the obstacle.

KMESHX(3,*iobst*) contains nx.
In order to check whether a point is in the obstacle or not, the obstacle is covered by a rectangular grid of size nx × ny × nz.

KMESHX(4,*iobst*) contains ny.

KMESHX(5,*iobst*) contains nz.

KMESHX(6,*iobst*) contains the memory management sequence number of array iobgrid. This array of size nx × ny × nz × leniobgrid, contains information about the rectangular grid that covers the obstacle in the following way.
Let iobgrid be a four-dimensional array iobgrid(1:nx, 1:ny, 1:nz, 1:leniobgrid).
The region covered is defined by xmin, xmax, ymin, ymax, zmin, zmax. The cell size is dx × dy× dz. Cell i, j, k refers to the co-ordinates:
(xmin+(i-1)dx,ymin+(j-1)dy),zmin+(k-1)dz × (xmin+idx,ymin+jdy,zmin+kdz).
iobgrid(i, j, k, l) may corresponds to:

iobgrid$(i, j, k, 1)$ may take one of the following values:

0 cell outside obstacle.

l>0 boundary part l crosses cell.

-1 cell completely in obstacle.

iobgrid$(i, j, k, 2...$*leniobgrid*$)$ = l1 >0 if there are more boundary parts l that crosses the cell i, j, k.

KMESHX(7,*iobst*) contains leniobgrid, defining the maximum value of the the fourth dimension parameter in iobgrid.

KMESHX(8,*iobst*) contains the memory management sequence number of array kmeshc_iobst. This array contains the topology of the obstacle surface, i.e. the local node numbers of the local elements in the surface.
Compare with KMESH part c.

KMESHX(9,*iobst*) contains the memory management sequence number of array coor_iobst. This array contains the coordinates of the nodes in the obstacle in the local sequence.
Compare with COOR.

KMESHX(10,*iobst*) contains INPELM, i.e. the number of nodes per element in the obstacle surface.

KMESHX(11,*iobst*) contains NELEM, i.e. the number of element in the obstacle surface.

KMESHX(12,*iobst*) contains NPOINT, i.e. the number of nodes in the obstacle surface.

RMESHX may be considered as a two dimensional array of size LENRL × NUMOBST with the following contents:
In R^2

RMESHX(1,*iobst***)** = xmin
RMESHX(2,*iobst***)** = xmax
RMESHX(3,*iobst***)** = ymin
RMESHX(4,*iobst***)** = ymax
RMESHX(5,*iobst***)** = dx
RMESHX(6,*iobst***)** = dy

In R^3

RMESHX(1,*iobst***)** = xmin
RMESHX(2,*iobst***)** = xmax
RMESHX(3,*iobst***)** = ymin
RMESHX(4,*iobst***)** = ymax
RMESHX(5,*iobst***)** = zmin
RMESHX(6,*iobst***)** = zmax
RMESHX(7,*iobst***)** = dx
RMESHX(8,*iobst***)** = dy
RMESHX(9,*iobst***)** = dz

part aa KMESH part aa is a very special array in the sense that it contains a lot of information that is already stored in the rest of KMESH. However, this information is edge and face oriented rather than node oriented. Furthermore, the type of storage differs somewhat from the usual storage, in order to simplify some programming parts.
In the sequel we shall use the name NMESH instead of KMESH part aa.
NMESH itself is stored in KMESH. It also contains memory management references so that actual information can be found in IBUFFR. Some parts of this array are only filled if they are actually used, so that the amount of storage may be very limited.
The starting address of NMESH is defined by KELMAA, hence the first position of NMESH is KMESH(KELMAA).
The length of NMESH is 50. Not all 50 positions are actually used.
These 100 positions in NMESH are defined as follows:

1-3 Not yet used
4 nedges: total number of edges.
5 nfaces: total number of faces.
6-9 Not yet used
10 nconnd: number of nodes connected
11 nconed: number of edges connected
12 nconfa: number of faces connected
13 nconel: number of elements connected
14 melpnd: max number of elements per node
15 melped: max number of elements per edge
16 melpfa: max number of elements per face
17 mfapnd: max number of faces per node
18 mfaped: max number of faces per edge

19 mfapel: max number of faces per element

20 medpnd: max number of edges per node

21 medpfa: max number of edges per face

22 medpel: max number of edges per element

23 mndped: max number of nodes per edge

24 mndpfa: max number of nodes per face

25 mndpel: max number of nodes per element

26 nspecp: number of points along an edge of a spectral element

27 medpcu: maximum number of edges in curve

28 mfapsu: maximum number of faces in surface

29 melpvo: maximum number of elements in volume

30 Not yet used

31 MMNMESHA memory management sequence number of part a. This part contains the face sequence numbers of the elements.

32 MMNMESHB memory management sequence number of part b. This part contains the edge sequence numbers of the faces.

33 MMNMESHC memory management sequence number of part c. This part contains the node numbers of which the edges consist.

34 MMNMESHD memory management sequence number of part d. This part contains the edge sequence numbers of the elements.

35 MMNMESHE memory management sequence number of part e. This part contains the node sequence numbers of the elements. This is an alternative for the combination KMESHc, KMESHg.

36 MMNMESHF memory management sequence number of part f. This part contains the element sequence numbers of the elements connected to the faces.

37 MMNMESHG memory management sequence number of part g. This part contains the node sequence numbers of the faces.

38 MMNMESHH memory management sequence number of part h. This part contains the element sequence numbers of the elements connected to the edges.

39 MMNMESHI memory management sequence number of part i. This part contains the face sequence numbers of the faces connected to the edges.

40 MMNMESHJ memory management sequence number of part j. This part contains the face sequence numbers of the faces of surfaces in 3D.

41 MMNMESHK memory management sequence number of part k. This part contains the face sequence numbers of the faces connected to the nodes.

42 MMNMESHL memory management sequence number of part l. This part contains the edge sequence numbers of the edges connected to the nodes.

43-50 Not yet used

The various parts of NMESH that are stored in IBUFFR are defined as follows:

part A This part contains the face sequence numbers of the elements.
The array is stored in IBUFFR with starting address IPNMESHA. IPNMESHA can be computed by IPNMESHA=iniget(MMNMESHA).
Array NMESHA is stored as follows:
The first NELEM positions contain the starting positions of the information for each element.
Hence IPSTART = NMESHA(ielem) contains the starting address of element ielem.

For each element 2 × the number of faces of that element (NFACE_ELM) plus 1 positions are used to store the information.
The first position contains NFACE_ELM.
The positions 2 to NFACE_ELM+1 contains the face sequence number of the faces of the element.
If the normal defined on the face (using the outer product of the first and second edge) is directed inwardly for the element, the face sequence number is provided with a minus sign.

The numbering of the faces is related to the local node numbering of Table 2.2.1 in the Users Manual as follows:

shape numbers 1 and 2 No face is defined

shape numbers 3 to 10 The element itself is the face

shape numbers 11 to 18 The faces are defined as follows

shape no	face no	local node numbers
11	1	1, 2, 4
18	2	2, 3, 4
	3	3, 1, 4
	4	1, 3, 2
12	1	1, 2, 3, 8, 10, 7
	2	3, 4, 5, 9, 10, 8
	3	5, 6, 1, 7, 10, 9
	4	1, 6, 5, 4, 3, 2
13	1	1, 2, 6, 5
17	2	3, 4, 8, 7
	3	2, 1, 4, 3
	4	5, 6, 7, 8
	5	1, 5, 8, 4
	6	2, 3, 7, 6
14	1	1, 2, 3, 12, 21, 20, 19, 10, 11
	2	9, 8, 7, 16, 25, 26, 27, 18, 17
	3	3, 2, 1, 4, 7, 8, 9, 6, 5
	4	19, 20, 21, 24, 27, 26, 25, 22, 23
	5	1, 10, 19, 22, 25, 16, 7, 4, 13
	6	3, 6, 9, 18, 27, 24, 21, 12, 15
15	1	1, 2, 3, 8, 10, 7, 12
16	2	3, 4, 5, 9, 10, 8, 13
	3	5, 6, 1, 7, 10, 9, 14
	4	1, 6, 5, 4, 3, 2, 11
16	1	1, 2, 3, 8, 10, 7, 12
	2	3, 4, 5, 9, 10, 8, 13
	3	5, 6, 1, 7, 10, 9, 14
	4	1, 6, 5, 4, 3, 2, 11

The positions NFACE_ELM+2 to 2 × NFACE_ELM+1 contain rotations that are necessary to express the elements uniquely into faces.
The rotations are defined in the following way:
Suppose you have a triangular face with nodes 1, 2 and 3. Rotations are defined in a right-handed way. So one rotation gives nodes 2, 3 and 1, two rotations 3, 2 and 1 and three rotations gives the original element.
For quadrilaterals we need 4 rotations to return to the original position.
The reason why we need this is that the face of an element does not have to coincide with a face as defined in the series of faces.

For example let face 17 have global node numbers 4, 9, 12 and 35. Let element 223 be a hexahedron with nodes 1, 3, 6, 7, 35, 4, 9 and 12. Then the fourth face of element 223 is face number 17 with 3 rotations.

So to find the i^{th} face corresponding to element j the following statements are used:

```
ipstart = nmesha(j)
nfaceelm = nmesha(ipstart)
ifacenr = nmesha(ipstart+j)
```

If an element is a surface element, NFACE_ELM is equal to 1 and the rotation is always 0.
If the element is a line element, NFACE_ELM is equal to 0.

part B This part contains the edge sequence numbers of the faces.
The array is stored in IBUFFR with starting address IPNMESHB. IPNMESHB can be computed by IPNMESHB=iniget(MMNMESHB).
Array NMESHB is stored as follows:
The first NFACES positions contain the starting positions of the information for each face.
Hence IPSTART = NMESHB(iface) contains the starting address of face iface.
For each face the number of edges of that face (NEDGE_FACE) plus 1 positions are used to store the information.
The first position contains NEDGE_FACE .
The positions 2 to NEDGE_FACE+1 contains the edge sequence number of the edges of the face.
If the edge is defined in reversed order, the edge sequence number is provided with a minus sign.
The edges of the faces are numbered as follows (shape numbers refer to Table 2.2.1 in the Users Manual): The numbering of the faces is related to the local node numbering of Table 2.2.1 in the Users Manual as follows:

shape numbers 1 and 2 The element itself is the edge

shape numbers 3 to 18 The faces are defined as follows

shape no	edge no	local node numbers
3	1	1, 2
10	2	2, 3
	3	3, 1
4	1	1, 2, 3
7	2	3, 4, 5
	3	5, 6, 1
5	1	1, 2
9	2	2, 3
	3	3, 4
	4	3, 1
6	1	1, 2, 3
	2	3, 4, 5
	3	5, 6, 7
	4	7, 8, 1
8	1	1, 2, 3, 4
	2	4, 5, 6, 7
	3	7, 8, 9, 1
11	1	1, 2
18	2	2, 4
	3	4, 1
	4	1, 3
	5	2, 3
	6	4, 3
12	1	1, 2, 3
15	2	3, 8, 10
16	3	10, 7, 1
	4	1, 6, 5
	5	3, 4, 5
	6	10, 9, 5
13	1	1, 2
17	2	2, 6
	3	6, 5
	4	5, 1
	5	4, 3
	6	3, 7
	7	7, 8
	8	8, 4
	9	1, 4
	10	2, 3
	11	6, 7
	12	5, 8
14	1	1, 2, 3
	2	3, 12, 21
	3	21, 20, 19
	4	19, 10, 1
	5	7, 8, 9
	6	9, 18, 27
	7	27, 26, 25
	8	25, 16, 7
	9	1, 4, 7
	10	3, 6, 9
	11	21, 24, 27
	12	19, 22, 25

Exactly the same structure to find faces corresponding to elements may be used to find

edges corresponding to faces.

part C This part contains the node numbers of which the edges consist.
The array is stored in IBUFFR with starting address IPNMESHC. IPNMESHC can be computed by IPNMESHC=iniget(MMNMESHC).
Array NMESHC is stored as follows:
The first NEDGES positions contain the starting positions of the information for each edge.
Hence IPSTART = NMESHC(iedge) contains the starting address of edge iedge.
For each edge the number of nodes of that edge (NNODE_EDGE) plus 1 positions are used to store the information.
The first position contains NNODE_EDGE .
The positions 2 to NNODE_EDGE+1 contains the node sequence number of the nodes of the edge.

The direction of an edge is defined as follows.
Suppose that an edge is defined from node_begin to node_end. The direction of the edge is such that the outer product of the normal of the face and the tangent of the face (defined as the vector from node_begin to node_end) is directed outward for the face with the lowest sequence number.

Exactly the same structure to find faces corresponding to elements may be used to find nodes corresponding to edges.

part D This part contains the edge sequence numbers of the elements.
The array is stored in IBUFFR with starting address IPNMESHD. IPNMESHD can be computed by IPNMESHD=iniget(MMNMESHD).
Array NMESHD is stored as follows:
The first NELEM positions contain the starting positions of the information for each element.
Hence IPSTART = NMESHD(ielem) contains the starting address of element ielem.
For each element the number of edges of that element (NEDGE_ELEM) plus 1 positions are used to store the information.
The first position contains NEDGE_ELEM .
The positions 2 to NEDGE_ELEM+1 contains the edge sequence number of the edges of the element.
Exactly the same structure to find faces corresponding to elements may be used to find edges corresponding to elements.

part E This part contains the node sequence numbers of the elements. This is an alternative for the combination KMESHc, KMESHg.
The array is stored in IBUFFR with starting address IPNMESHE. IPNMESHE can be computed by IPNMESHE=iniget(MMNMESHE).
Array NMESHE is stored as follows:
The first NELEM positions contain the starting positions of the information for each element.
Hence IPSTART = NMESHE(ielem) contains the starting address of element ielem.
For each element the number of nodes of that element (NODE_ELEM) plus 1 positions are used to store the information.
The first position contains NODE_ELEM .
The positions 2 to NODE_ELEM+1 contains the node sequence number of the nodes of the element.
Exactly the same structure to find faces corresponding to elements may be used to find nodes corresponding to elements.

part F This part contains the element sequence numbers of the elements connected to the faces.
The array is stored in IBUFFR with starting address IPNMESHF. IPNMESHF can be computed by IPNMESHF=iniget(MMNMESHF).

Array NMESHF is stored as follows:
The first NFACES positions contain the starting positions of the information for each element.
Hence IPSTART = NMESHF(ielem) contains the starting address of face iface.
For each face the number of elements connected to that face (NFACE_ELEM) plus 1 positions are used to store the information.
The first position contains NFACE_ELEM .
The positions 2 to NFACE_ELEM+1 contains the element sequence number of the elements corresponding to the face.
Exactly the same structure to find faces corresponding to elements may be used to find elements corresponding to faces.

part G This part contains the node sequence numbers of the faces.
The array is stored in IBUFFR with starting address IPNMESHG. IPNMESHG can be computed by IPNMESHG=iniget(MMNMESHG).
Array NMESHG is stored as follows:
The first NFACES positions contain the starting positions of the information for each node.
Hence IPSTART = NMESHG(nodp) contains the starting address of node nodp.
For each face the number of nodes in that face (NFACE_NODE) plus 1 positions are used to store the information.
The first position contains NFACE_NODE .
The positions 2 to NFACE_NODE+1 contains the node sequence number of the nodes in the face.
Exactly the same structure to find faces corresponding to elements may be used to find nodes in the faces.

part H This part contains the element sequence numbers of the elements connected to the edges.
The array is stored in IBUFFR with starting address IPNMESHH. IPNMESHH can be computed by IPNMESHH=iniget(MMNMESHH).
Array NMESHH is stored as follows:
The first NEDGES positions contain the starting positions of the information for each edge.
Hence IPSTART = NMESHH(iedge) contains the starting address of edge iedge.
For each edge the number of elements connected to that edge (NEDGE_ELEM) plus 1 positions are used to store the information.
The first position contains NEDGE_ELEM .
The positions 2 to NEDGE_ELEM+1 contains the element sequence number of the elements corresponding to the edge.
Exactly the same structure to find faces corresponding to elements may be used to find elements corresponding to edges.

part I This part contains the face sequence numbers of the faces connected to the edges.
The array is stored in IBUFFR with starting address IPNMESHI. IPNMESHI can be computed by IPNMESHI=iniget(MMNMESHI).
Array NMESHI is stored as follows:
The first NEDGES positions contain the starting positions of the information for each edge.
Hence IPSTART = NMESHI(iedge) contains the starting address of edge iedge.
For each edge the number of faces connected to that face (NEDGE_FACE) plus 1 positions are used to store the information.
The first position contains NEDGE_FACE .
The positions 2 to NEDGE_FACE+1 contains the face sequence number of the faces corresponding to the edge.
Exactly the same structure to find faces corresponding to elements may be used to find faces corresponding to edges.

part J This part contains the face sequence numbers of the faces of the surfaces in 3D.
The array is stored in IBUFFR with starting address IPNMESHJ. IPNMESHJ can be computed by IPNMESHJ=iniget(MMNMESHJ).
Array NMESHJ has length NUMELEMS (number of elements in the surfaces)
The face numbers are stored in exactly the same sequence as the elements in KMESH part n.

part K This part contains the face sequence numbers of the faces connected to the nodes.
The array is stored in IBUFFR with starting address IPNMESHK. IPNMESHK can be computed by IPNMESHK=iniget(MMNMESHK).
Array NMESHK is stored as follows:
The first NPOINT positions contain the starting positions of the information for each node.
Hence IPSTART = NMESHK(nodp) contains the starting address of node nodp.
For each node the number of faces connected to that node (NNODE_FACE) plus 1 positions are used to store the information.
The first position contains NNODE_FACE .
The positions 2 to NNODE_FACE+1 contains the face sequence number of the faces corresponding to the node.
Exactly the same structure to find faces corresponding to elements may be used to find faces corresponding to nodes.

part L This part contains the edge sequence numbers of the edges connected to the nodes.
The array is stored in IBUFFR with starting address IPNMESHL. IPNMESHL can be computed by IPNMESHL=iniget(MMNMESHL).
Array NMESHL is stored as follows:
The first NPOINT positions contain the starting positions of the information for each node.
Hence IPSTART = NMESHL(nodp) contains the starting address of node nodp.
For each node the number of edges connected to that node (NNODE_EDGE) plus 1 positions are used to store the information.
The first position contains NNODE_EDGE .
The positions 2 to NNODE_EDGE+1 contains the edge sequence number of the edges corresponding to the node.
Exactly the same structure to find faces corresponding to elements may be used to find edges corresponding to nodes.

part ab KMESH part ab is used to store information about elements, curves, surfaces and volumes in a form that is very suited for programming. KMESH part ab itself is stored in KMESH. It also contains memory management references so that actual information can be found in IBUFFR.
The same type of storage as in KMESH part aa is used.
The starting address of KMESH part ab is defined by KELMAB, hence the first position of KMESH part ab is KMESH(KELMAB).
The length of KMESH part ab is 10. Not all 10 positions are actually used.
These 20 positions in KMESH part ab are defined as follows:

1 MMELGRPS: memory management sequence number of information about the element groups.

2 MMINFVOLMS: memory management sequence number of information about the volumes

3 MMINFSURFS: memory management sequence number of information about the surfaces

4 MMINFCURVS: memory management sequence number of information about the curves

5 MMINFUSPNT: memory management sequence number of information about the user points

6 MMSUBVOLMS: memory management sequence number of information about subvolumes of volumes

7 MMSUBSURFS: memory management sequence number of information about subsurfaces of surfaces

8 MMSUBCURVS: memory management sequence number of information about subcurves of curves

9 MMELSVOLMS: memory management sequence number of information about volumes elements inside a volume.

10 MMELSSURFS: memory management sequence number of information about faces inside a surface.

11 MMELSCURVS: memory management sequence number of information about edges inside a curve.

12 MMELSUSPNT: memory management sequence number of information about elements inside a user point.

13 MMDATVOLMS: memory management sequence number of information about surfaces corresponding to the volumes

14 MMDATSURFS: memory management sequence number of information about curves corresponding to the surfaces

15 MMDATCURVS: memory management sequence number of information about user points corresponding to the curves

16-20 Not yet used

The various parts of KMESHab that are stored in IBUFFR are defined as follows:

part ELGRPS This part contains ordered information about the element groups. It is a compilation of parts A, B, F and U.
The array is stored in IBUFFR with starting address IPELGRPS. IPELGRPS can be computed by IPELGRPS=iniget(MMELGRPS).
Array ELGRPS is stored as follows:
The first NELGRP positions contain the starting positions of the information for each element.
Hence IPSTART = ELGRPS(ielgrp) contains the starting address of element group ielgrp.
For each element group the amount of information stored is defined as NINF_elgrp. At this moment NINF_elgrp is equal to 8 but this number may be changed in the future.
For each element group the following information is stored.

Pos. 1 NINF_elgrp (=8)

Pos. 2 Number of nodes per element (KMESH part a)

Pos. 3 Number of boundary nodal points per element (KMESH part b)

Pos. 4 Shape number of elements (KMESH part f)

Pos. 5 Number of extra points in a side or edge of a spectral element (KMESH part u)

Pos. 6 Number of extra internal points in a surface or face of a spectral element (KMESH part u)

Pos. 7 Number of extra internal points in a volume spectral element if the boundary points are excluded. (KMESH part u)

Pos. 8 Parameter that indicates in which way the extra points have to be positioned on the sides and as a consequence internally. Possible values:

0: subdivision is equidistant.

1: subdivision is in accordance with a Gauss-Legendre-Lobatto polynomial.

Pos. 9 number of elements in group

So to find the number of nodes corresponding to element group j the following statements are used:

```
ipstart = elgrps(j)
inpelm = elgrps(ipstart+j)
```

part INFVOLMS This part contains ordered information about the volumes. It contains information from IINPUT.
The array is stored in IBUFFR with starting address IPINFVOLMS. IPINFVOLMS can be computed by IPINFVOLMS=iniget(MMINFVOLMS).
Array INFVOLMS is stored as follows:
The first NVOLMS positions contain the starting positions of the information for each volume.
Hence IPSTART = INFVOLMS(icurnr) contains the starting address of volume icurnr.
For each volume the amount of information stored is defined as NINF_volumes. At this moment NINF_volumes is equal to 8 but this number may be changed in the future.
For each volume the following information is stored.

Pos. 1 NINF_volumes (=9)

Pos. 2 IVOLM (type of volume, i.e. by which generator created, see mesh generator)
Possible values:

1. BRICK
2. USER
3. INPUT
4. PIPE
5. GENERAL
6. TRANSLATE
7. ROTATE
8. REFLECT

Pos. 3 ISHAPE (shape of the elements, see mesh generator)

Pos. 4-9 The contents of these positions depend on the value of ISURF.
If fewer quantities are stored, the rest of the positions is undefined.

IVOLM = 1 N, M, L

IVOLM = 2 NELEM, NPOINT

IVOLM = 3 FILE-REFERENCE-NUMBER, NELEM, NPOINT

IVOLM = 4,5 -

IVOLM = 6, 7, 8 Volume number of volume to be copied, followed by volume number of originating volume (recursively)

Pos. 10 number of elements in volume

part INFSURFS This part contains ordered information about the surfaces. It contains information from parts M and IINPUT.
The array is stored in IBUFFR with starting address IPINFSURFS. IPINFSURFS can be computed by IPINFSURFS=iniget(MMINFSURFS).
Array INFSURFS is stored as follows:
The first NSURFS positions contain the starting positions of the information for each surface.
Hence IPSTART = INFSURFS(icurnr) contains the starting address of surface icurnr.
For each surface the amount of information stored is defined as NINF_surfaces. At this moment NINF_surfaces is equal to 8 but this number may be changed in the future.
For each surface the following information is stored.

Pos. 1 NINF_surfaces (=9)

Pos. 2 ISURF (type of surface, i.e. by which generator created, see mesh generator)
Possible values:

1. RECTANGLE
2. GENERAL
3. TRIANGLE
4. QUADRILATERAL
5. MESHUS
6. INPUT
7. SURFACES OF SURFACES
8. TRANSLATE
9. ROTATE
10. PIPESURFACE
11. ORDERED SURFACES
12. ORDERED PIPE SURFACES
13. COONS
14. SIMILAR
15. PARSURF
16. REFLECT
17. RECTANGLE with smoothing
18. ISOPAR

Pos. 3 ISHAPE (shape of the elements, see mesh generator)

Pos. 4-9 The contents of these positions depend on the value of ISURF.
If fewer quantities are stored, the rest of the positions is undefined.

ISURF = 1,17 N, M

ISURF = 2,15,18 -

ISURF = 5 NELEM, NPOINT

ISURF = 6 FILE-REFERENCE-NUMBER, NELEM, NPOINT

ISURF = 7 Number of external boundaries, number of internal boundaries

ISURF = 8,9,14,16 surface number of surface to be copied, followed by surface number of originating surface (recursively)

ISURF = 10 Number of boundaries, C1, C2, C3, C4, N, M

ISURF = 11 Number of external boundaries, number of internal boundaries, N, M

ISURF = 12 Number of boundaries, number of C1, C2, C3, C4, N, M

Pos. 10 number of faces in surface

part INFCURVS This part contains ordered information about the curves. It contains information from parts M and IINPUT.
The array is stored in IBUFFR with starting address IPINFCURVS. IPINFCURVS can be computed by IPINFCURVS=iniget(MMINFCURVS).
Array INFCURVS is stored as follows:
The first NCURVS positions contain the starting positions of the information for each curve.
Hence IPSTART = INFCURVS(icurnr) contains the starting address of curve icurnr.
For each curve the amount of information stored is defined as NINF_curves. At this moment NINF_curves is equal to 8 but this number may be changed in the future.
For each curve the following information is stored.

Pos. 1 NINF_curves (=9)

Pos. 2 icurve, type of curve see Section 3.2

Pos. 3 ishape, shape of elements along curve

Pos. 4 nelm, number of elements along curve

Pos. 5 iratio

Pos. 6 odd

Pos. 7 itype_spline

Pos. 8 ialpha

Pos. 9 0 if curve is outer curve, 1 if it is an inner curve

Pos. 10 number of edges in curve

part SUBVOLMS This part contains all subvolumes of the volumes. Since at this moment volumes of volumes have not yet been implemented, this part does not yet exist.

part SUBSURFS This part contains all subsurfaces of the surfaces, hence it contains the same information as part q, however, organized in a slight different form.
The first NSURFS positions contain the starting address of surface isurnr. For each surface 1 plus the number of subsurface positions are used to store the information.
The first position contains the number of subsurfaces, the following positions the sequence numbers of these subsurfaces.
Hence exactly the same type of storage is used as in KMESH part aa (see part A).

part SUBCURVS Is identical to the part SUBSURFS, but now with respect to the curves.

part ELSVOLMS This part contains all element sequence numbers of the volume elements corresponding to the volumes.
The first NVOLMS positions contain the starting address of volume ivolnr. For each volume 1 plus the number of elements in that volume positions are used to store the information.
The first position contains the number of elements, the following positions the sequence numbers of these elements.
Hence exactly the same type of storage is used as in KMESH part aa (see part A).

part ELSSURFS Contains the face numbers in the surfaces in the following way:

Pos. 1 to nsurfs Starting addresses ipstartsurnr of the surfaces

Pos. ipstartsurnr Number of faces n in the surface times 2

Pos. ipstartsurnr+1 to ipstartsurnr+n face numbers provided with sign

Pos. ipstartsurnr+n+1 to ipstartsurnr+2n rotation of face numbers to fit into kmesh part n.

part ELSCURVS Is identical to the part ELSVOLMS, but now with respect to edges in the curves.

part ELSUSPNT Is identical to the part ELSVOLMS, but now with respect to the user points.

part DATVOLMS Contains a list of all surfaces corresponding to the volumes. Exactly the same type of storage is used as in KMESH part aa, which means that first nvolms starting addresses are given followed by the actual information. The actual information per volume consists of the number of surfaces corresponding to the volume, followed by the surface sequence numbers.

part DATSURFS Is identical to the part DATVOLMS, but now with respect to the surfaces.

part DATCURVS memory management sequence number of information Is identical to the part DATVOLMS, but now with respect to the curves.

part y KMESH part y is only needed for the generation of a mesh for the unstructured finite volume flow solver. KMESH part y can be generated by routine `iniactmk`.
The use of KMESH part y is at this moment restricted to 2d meshes that consist solely out of triangles. In KMESH part y a terminology different from the common finite-element terminology is used.
In particular:

- Edges will be called faces in this part.

- The triangles, called faces in the finite-element world, are now called cells. Thus, in 2D problems the cells are equivalent to the elements in the finite-element framework.
- The quantity `nedges` is called `nfaces`. Note that `nfaces` = kmesh(kmesh(48)+3).

The length of KMESH part y is 30. Not all 30 positions are actually used.
These 30 positions in KMESH part y are defined as follows:

1 Not yet used

2 = `nfacesi`, where `nfacesi` is the number of internal faces in the mesh.

3 = `nfacesir`, where `nfacesir` is the number of real internal faces in the mesh.

4 = `nelemi`, where `nelemi` is the number of internal cells.

5-20 Not yet used

21 : memory management sequence number referring to array `face`.

22 : memory management sequence number referring to array `iconcell`.

23 : memory management sequence number referring to array `iconface`.

24 : memory management sequence number referring to array `lengthf`.

25 : memory management sequence number referring to array `areafv`.

26-30 Not yet used

KMESHY(2) - KMESHY(4)
The elements 2, 3 and 4 of KMESHY, together with `nelem` = kmesh(9), `npoint` = kmesh(8), `nedges` = `nfaces` = kmesh(kmesh(48)+3) and the coordinates of the `npoint` vertices (memory management sequence number is kmesh(23)), determine the general set-up of a 2D mesh as required for the unstructured finite volume flow solver.

The elements (often called cells) that are present in the 2D mesh, must be triangles. There are `nelemi` internal cells (i.e. cells that have no faces at the boundary) and (`nelem` - `nelemi`) boundary cells (i.e. cells that have at least one boundary face).
We distinguish between boundary faces and internal faces. Boundary faces are faces that are part of the boundary, and internal faces are faces that are not part of the boundary. The number of boundary faces follows from (`nfaces` - `nfacesi`). We distinguish between two kinds of internal faces: real internal faces and quasi internal faces. A quasi internal face is an internal face that is part of a boundary cell. A real internal face is an internal face that is not part of a boundary cell. This means, for example, that every face that makes part of a boundary cell is either a boundary or a quasi internal face, but never a real internal face. The number of quasi internal faces equals (`nfacesi` - `nfacesir`).
Two useful relations are: `nelem` = (`nfaces` + `nfacesi`)/3 and (for a mesh without holes) `nelem` + `npoint` = `nfaces` + 1.

KMESHY(21) is the memory management sequence number of array `face` in array IBUFFR. Array `face` is an integer (6 × `nfaces`)-array, with the following contents:

- `face`(1,e) and `face`(2,e) contain the vertex numbers that define face e.
- `face`(3,e) and `face`(5,e) contain the cell numbers of the cells adjacent to face e.
- `face`(4,e) and `face`(6,e) contain the local face numbers of respectively cells `face`(3,e) and `face`(5,e). The local face number is the column-position (1, 2 or 3) in which the face is stored for the corresponding cell in array `cellfv`.

Array `cellfv` is a (3 × `nelem`)-array containing the face numbers connected to each cell. This array can easily be constructed when we have nmesh part b and restrict ourselves to triangular cells. To show this by means of an example, assume that `nelem` = 3, and that nmesh part b is given by:

```
4   8  12   3  -5   9  -2   3  10  -2  11   3  -7  -9 -11.
```

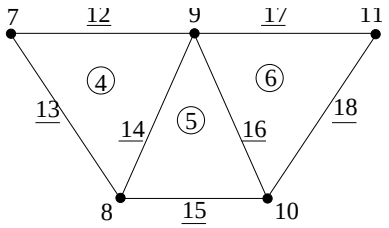


Figure 24.2.5: Example of a mesh. Cell numbers are in circles, face numbers are underlined, and the other numbers are vertex identification numbers.

Then array `cellfv` is given by: cell 1: `-5 9 -2`
cell 2: `10 -2 11`
cell 3: `-7 -9 -11`.
In the computer-code array `cellfv(3,nelem)` can be obtained in a subroutine using the following commands:

- `call iniactmk(ibuffr,2,32,kmesh,dummy,mmnmeshb`, where `mmnmeshb` is memory management number;
- `ipnmeshb = iniget(mmnmeshb)`, where `ipnmeshb` is pointing to the starting address of array nmesh part b;
- write `ibuffr(ipnmeshb+nelem)` in the CALL-statement to the subroutine where `cellfv` is required;
- in the subroutine, write `cellfv(0:3,nelem)` in the parameter-statement, so that `cellfv(1,*)`, `cellfv(2,*)` and `cellfv(3,*)` refer to the face-numbers (`cellfv(0,*)` stands for the the number of faces per cell, which is always 3 in this case).

The following conventions are made in storing array `face`:

- `face`(3,e) < `face`(5,e) always.
- the cell with cell number `face`(3,e) must always lie 'left' of the vector defined by (`face`(2,e) - `face`(1,e)), and consequently the cell with cell number `face`(5,e) lies always 'right' of this vector. The normal vector of (`face`(2,e) - `face`(1,e)) that is obtained after rotating this vector by $\pi/2$ radians in the counterclockwise direction, is pointing towards the cell that lies 'left'. The definition of what is 'right' now is trivial. Note that due to this convention we cannot interchange `face`(1,e) and `face`(2,e).

In the case that face e is a boundary face, `face`(5,e) is put to (`nelem` + 1), and `face`(6,e) is put to zero. Due to this choice and the conventions mentioned above, at the boundary faces the normal is always pointing outwards the domain.

To illustrate the use of arrays `cellfv` and `face`, consider Figure 24.2.5. Array `cellfv` has the following contents (the face numbers may be permutated in a cyclic manner):
cell 4: `-12 13 14`
cell 5: `-14 15 -16`
cell 6: `18 -17 16`.
Assuming that faces 12, 13, 15, 17 and 18 are boundary faces and that `nelem` = 6, array `face` has the following contents:

```
face 12:  9   7   4   1   7   0
face 13:  7   8   4   2   7   0
face 14:  8   9   4   3   5   1
face 15:  8  10   5   2   7   0
face 16: 10   9   5   3   6   3
face 17: 11   9   6   2   7   0
```

face 18: `10 11 6 1 7 0`.

KMESHY(22) is the memory management sequence number of array `iconcell` in array IBUFFR. Array `iconcell` is a (1 × `nelem`)-array of which the first `nelemi` elements are the cell numbers of the internal cells(, and consequently the next (`nelemi` + 1) to `nelem` elements are the cell numbers of the boundary cells). Note that this array is a permutation of array `cellfv`.

KMESHY(23) is the memory management sequence number of array `iconface` in array IBUFFR. Array `iconface` is a (2 × `nfaces`)-array. The first `nfaces`-elements have the following meaning: the first `nfacesir` elements are the face numbers of the real internal faces; the next elements, from (`nfacesir` + 1) to `nfacesi`, are the face numbers of the quasi internal faces, and the next elements, running from (`nfacesi` + 1) to `nfaces`, are the face numbers of the boundary faces). Note that this array is a permutation of the faces in array `face`.
The second part of array `iconface`, i.e. the elements from (`nfaces`+1) to 2*`nfaces`, is the inverse of the first part of array `iconface`. In the computer code this second part is often called `convarray`. This array satisfies: `convarray(iconface(i)) = i`.
In short, in the first part of array `iconface` a clear separation is made between the different types of faces, i.e. face `j` is a boundary, real internal or quasi internal face depending on the value of `i` that satisfies: `j = iconface(i)`. Furthermore, knowing face-number `j`, its position in array `iconface` then follows from: `i = convarray(j)`.

KMESHY(24) is the memory management sequence number of array `lengthf` in array BUFFER. Array `lengthf` is a real (1 × `nfaces`)-array containing the length of each face.

KMESHY(25) is the memory management sequence number of array `areafv` in array BUFFER. Array `areafv` is a real (1 × `nelem`)-array containing the area of each cell.

part z KMESH part z is only used in case the mesh is changed by subroutine ADAPBOUN or one of its related subroutines. It indicates for each curve whether it is changed or not. This part is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KELMZ as sequence number.
Length: NCURVS.

Contents: Part z is an integer array of length NCURVS.
KMESHZ(i) corresponds to curve i.
The following values of KMESHZ(i) are possible:

0 curve i has not been changed,

1 curve i does not belong to the free boundary, but since it has common points with the free boundary it may be changed.

2 curve i belongs to the free boundary and the nodes at the boundary may be redistributed.

3 curve i belongs to the free boundary and the nodes at the boundary may not be redistributed.

part ac KMESH part ac is only filled if an explicit call to subroutine mshsortelm is carried out. Only in a limited number of cases (like for example when using fictitious unknowns) this call is activated automatically.
If KMESH(51) > 0, then KMESH part ac has been filled.
KMESH ac contains information about the distribution of the nodes with respect to a rectangular grid.
The region defined by (xmin,xmax) × (ymin,ymax) × (zmin,zmax) is provided with a rectangular grid of nx × ny× nz cells with equal size per direction. So point (x,y,z) is lying in the cell ($\frac{int(x-xmin)}{\Delta x}+1$, $\frac{int(y-ymin)}{\Delta y}+1$, $\frac{int(z-zmin)}{\Delta z}+1$), where int is the largest integer that is not larger than the argument and $\Delta x = \frac{xmax-xmin}{nx}$ etc.

KMESH_ac consists of a real part RMESH_ac and an integer part also called KMESH_ac. The real part is stored in the first positions of the complete array, the integer part immediately follows that array.

RMESH_ac is a double precision array of length (NDIM+2) × NSPLIT, with NDIM the dimension of the space (2 or 3) and NSPLIT the number of disjoint parts the mesh consists of. For each disjoint part in R^2 the reals (xmin,xmax,ymin,ymax) are stored sequentially

The integer part KMESH_ac consists of a number of subparts.
kmeshac is organized as follows:
Sub-part 1: length (NDIM+2) × NSPLIT.
For each disjoint part NDIM+2 positions:

Pos. 1 (NNODES) number of nodes in part

pos. 2 to NDIM+1 NX,NY,NZ i.e. number of cells in each direction

pos. NDIM+2 starting address with respect to sub-part 2

For each disjoint part we have a separate sub-part 2 and sub-part 3.
Sub-part 2 of length 1+NX × NY × NZ contains starting address of nodes in sub-part 3 for each cell.
So in R^2 the number of nodes in cell (i,j) is kmeshac_2(k+1)-kmeshac_2(k), with k=(i-1)ny+j.
Sub-part 3 of length NNODES contains the node numbers in the sequence defined by part 2.
With respect to the use of array KMESH ac, consider the subroutines: PRFINDELM2 and PRFINDELM3.

part ad KMESH part ad is only filled if an explicit call to subroutine MSHFILLKMESHAD. is carried out or if subroutine INIACTMK (22.8.1) is called with ichoice=1 and ipos=52.
With respect to the use of array KMESH ad, consider the subroutines:
At this moment KMESH part ad is stored in KMESH from position KELMAD.
It uses 10 positions with the following contents:

Pos. 1 mmipointobst: memory management sequence number of array ipointobst
In order to get the value of mmipointobst and to activate or create this array use subroutine iniactmk:

```
call iniactmk ( ibuffr, 5, 1, kmesh, kmesh, mmipointobst )
```

Pos. 2 mmielemobst: memory management sequence number of array ielemobst
In order to get the value of mmielemobst and to activate or create this array use subroutine iniactmk:

```
call iniactmk ( ibuffr, 5, 2, kmesh, kmesh, mmielemobst )
```

Pos. 3 mmicrossobst: memory management sequence number of array icrossobst
In order to get the value of mmicrossobst and to activate or create this array use subroutine iniactmk:

```
call iniactmk ( ibuffr, 5, 3, kmesh, kmesh, mmicrossobst )
```

Pos. 4 mmrcrossobst: memory management sequence number of array rcrossobst
In order to get the value of mmrcrossobst and to activate or create this array use subroutine iniactmk:

```
call iniactmk ( ibuffr, 5, 4, kmesh, kmesh, mmrcrossobst )
```

Pos. 5 iseqobst: sequence number of the obstacle.
When the obstacle is created iseqobst gets the value 0.
Each time the obstacle is moved iseqobst is raised.

Pos. 6 iseqipoint: sequence number of the array ipointobst.
Each time this array is created iseqipoint gets the value iseqobst.
In this way it can be checked if ipointobst can be reused or must be recreated.

Pos. 7 iseqielem: sequence number of the array ielemobst.
Each time this array is created iseqielem gets the value iseqobst.
In this way it can be checked if ielemobst can be reused or must be recreated.

Pos. 8 iseqicross: sequence number of the array icrossobst.
Each time this array is created iseqicross gets the value iseqobst.
In this way it can be checked if icrossobst and rcrossobst can be reused or must be recreated.

Pos. 9 mmobstfaces: Memory management sequence number of array `OBSTACLE_FACES` containing the face or edge numbers corresponding to the obstacle.
In order to get the value of mmobstfaces and to activate or create this array use subroutine iniactmk:

```
call iniactmk ( ibuffr, 5, 9, kmesh, kmesh, mmobstfaces )
```

Pos. 10 Not yet used, equal to 0

The subarrays stored in IBUFFR, have the following meaning

ipointobst Array containing information about the nodes in an obstacle.
The first numobst (KMESH(42)) positions contain the starting positions of each obstacle sequentially. Hence IPOINTOBST(iobst) contains the starting position of obstacle iobst in array IPOINTOBST.
Per obstacle the first four positions contain

1. The number of nodes inside the obstacle that are only in elements that are completely inside the obstacle.
   See Users Manual Section 2.2: IN_INNER_OBSTACLE.
2. The number of nodes inside the obstacle that remain but are not on the boundary of the obstacle.
   See Users Manual Section 2.2: IN_BOUN_OBSTACLE.
3. The number of nodes on the boundary of the obstacle.
   See Users Manual Section 2.2: ON_BOUN_OBSTACLE.
4. The number of nodes on the boundary of the obstacle that are only in elements that are completely inside the obstacle.
   See Users Manual Section 2.2: INON_BOUN_OBSTACLE.

These positions are followed by the node numbers in the sequence defined by the first four positions. So first the nodes completely inside the obstacle and so on.

ielemobst Array containing information about the elements in an obstacle.
The first numobst (KMESH(42)) positions contain the starting positions of each obstacle sequentially. Hence IELEMOBST(iobst) contains the starting position of obstacle iobst in array IELEMOBST.
Per obstacle the first three positions contain

1. The number of elements that are completely inside the obstacle, i.e. all nodes are inside the obstacle.
2. The number of elements that are not completely inside the obstacle, but with all vertices inside the obstacle.
3. The number of elements that are partly inside the obstacle.

These positions are followed by the element numbers in the sequence defined by the first three positions.

icrossobst array containing information about the cross section of obstacles with edges.
The storage of icrossobst is as follows:

Pos. 1 to numobst (number of obstacles): icrossobst(iobst) = istartobst (starting address of obstacle iobst in array.

Per obstacle from position istartobst:
Number of cross sections, followed by 2 positions for each cross section consisting of the edge sequence number and a reference to rcrossobst.

This part is followed by the number of elements that is intersected by the obstacle followed by for each element: element number, number of intersections and relative sequence numbers of intersections repeatedly). These relative sequence numbers refer to the list of intersections stored for that particular obstacle.

rcrossobst Real array corresponding to icrossobst.
For each cross section a real factor f is stored defining the intersection of the edge with the obstacle in the following way:
$x_{\text{intersect}} = x_1 + f\ (x_2 - x_1)$
where x_1 and x_2 are the first and last point of the edge respectively

obstacle_faces This array contains the edge (R^2) or face (R^3) numbers of the boundary of the obstacle.
The storage is as follows:
Positions 1 to numobst: starting addresses of the obstacle with sequence number iobst.
For each obstacle: number of edges or faces of the boundary followed by edge or face numbers.

For example if one wants to have all the faces of the first obstacle, the following statements are required:

```
ipstart = obstacle_faces(1)
number_of_faces = obstacle_faces(ipstart)
do i = 1, number_of_faces
   i_th_face = obstacle_faces(ipstart+i)
end do
```

Part blockinfo Array blockinfo is only used in case of parallel computing.
It contains global block information for block number `blocknr`. In the future the description may be changed.
KMESH(54) contains the memory management sequence number of array blockinfo. Let `blocknr` be the present block.
It has the following contents:

Pos. 1 m (number of neighbor blocks)

Pos. 2*i block number `iblock` of i-th neighbor of `blocknr`.

Pos. 2*i+1 starting address of i-th neighbor `iblock` in array transinfo

Pos. 2m+1+i starting address of extra information of i-th neighbor `iblock` in array BLOCKINFO. This starting address is at least equal to 3m+2.

Pos. istart where istart is the starting address referred to in 2m+1+i is the first position with extra information. This extra information starts with the number of neighbor blocks m_i of block `iblock` that are also neighbor of the present block and contains extra information in transportinfo. This is followed by 2 m_i positions containing the sequence number of that neighbor block `ineigh` followed by a starting address in array transportinfo.

Part transportinfo Array transportinfo is only used in case of parallel computing.
It contains information about the transport of unknowns between neighboring blocks.
The starting address of each neighbor block is stored in array blockinfo.
KMESH(53) contains the memory management sequence number of array transportinfo.
It has the following contents:
For each neighbor block the following information is stored:

Pos. 1 number of nodes n on common side of the present block `blocknr` and the neighbor block `iblock`.

Pos. 2 ... n+1 local node numbers in this common side.

In the second part information of the intersection of neighbors of neighbors and neighbors of the present block `blocknr` as mentioned in BLOCKINFO is stored. The starting address can be found in BLOCKINFO
For each neighbor block iblock of block blocknr the following information is stored:

Pos. 1 number of nodes n_{ij} on the interface of the blocks `blocknr` and `iblock` that are also neighbor of the block `ineigh`. Hence these points do not belong to block `ineigh`.
For a definition, see blockinfo.

Pos. 2 ... n_{ij}+1 local node numbers in this intersection that are neighbor of the block `iblock` in increasing node sequence.

Part ielgrp_nodes Array containing all nodes corresponding to the element groups.
This array is stored in IBUFFR and has memory management sequence number KELMAE stored in KMESH(55).
It may be activated and created by a call to subroutine INIACTMK:

```
call iniactmk ( ibuffr, 1, 55, kmesh, kmesh, mmielgrp_nodes )
```

It has the following contents:

Pos. 1 to nelgrp starting addresses istart_ielgrp

Pos. istart_ielgrp: nnodes (number of nodes in element group ielgrp)

Pos. istart_ielgrp+1 to istart_ielgrp+nnodes: node numbers

Part node_elgrps Array containing all element groups corresponding to the nodes.
This array is stored in IBUFFR and has memory management sequence number KELMAF stored in KMESH(56).
It may be activated and created by a call to subroutine INIACTMK:

```
call iniactmk ( ibuffr, 1, 56, kmesh, kmesh, mmnode_elgrps )
```

It has the following contents:

Pos. 1 to npoint if negative -starting addresses istart_node.
if positive element group number'

Pos. istart_node: number of element groups (>1)

Next positions element group numbers

Part volume_surfs Array containing all surfaces corresponding to the volumes.
This array is only filled in case NVOLMS > 0. It has the following contents:

Pos. 1 to nvolms starting addresses for volume istart_vol in Part 2.
i

Pos. istart_vol: number of surfaces coupled to volume.

Next positions surface sequence numbers.
Only singe surfaces are stored, composite surfaces are split into single ones.

So to find all surfaces corresponding to volume ivolnr, use the following statements:

```
ipstart = volume_surfs(ivolnr)
number_of_surfaces = volume_surfs(ipstart)
do i = 1, number_of_surfaces
   surface_number = volume_surfs(ipstart+i)
end do
```

Part surfnodes Array containing the number of nodes and node numbers of surfaces.
The array is stored as follows:

Pos. 1 ... nsurfs Part 1: contains the starting addresses of each surface in part 2.

Part 2 Contains for each surface the number of nodes followed by node numbers.

Part outercurvs Array containing the curve numbers of the curves defining the outer boundary (R^2) or the surfaces defining the outer boundary (R^3).
In (R^2) the information is stored in the following way:

Pos. 1 nparts Number of parts the outer boundary consists of

Pos. 2-1+nparts Starting addresses of each part

Per part (first address: ioutercurvs(1+i)

Pos. 1 nouter Number of curves in part followed by the curve numbers (provided with a sign) These curve numbers are consecutive, which means the end of one curve is the start of the new one The part itself is closed.

In (R^3) the information is stored in the following way:

Pos. 1 nparts Number of parts the outer boundary consists of

Pos. 2-1+nparts Starting addresses of each part

Per part (first address: ioutersurfs(1+i)

Pos. 1 nouter Number of surfaces in part followed by the surface numbers (provided with a sign) The part itself is closed.

Part normals Array containing the normals on the surfaces in R^3, or the normals on the curves in R^2.
In (R^2) array normals has size 2 × LENGM3.
normal(i,k) contains the i^{th} component of the k^{th} node as stored in KMESHM part 3. So exactly the same sequence as in KMESHM part 3 is used. Normals in a point are defined as mean value of the normals of the two edges ending in that point.

In (R^3) array normals has size 3 × N, where N is the number of nodes stored in array surfnodes.
normal(i,k) contains the i^{th} component of the k^{th} node as stored in surfnodes. So exactly the same sequence as in array surfnodes is used. Normals in a point are defined as mean value of the normals of the elements containing that point.
Array normals is especially meant for parallel computation where the information of some neighbor elements may be missing since these elements are part of an other subdomain.

Part CURVE_USERP Array containing the user point numbers corresponding to the curves.
The storage of this array is as follows:

- curveusp(1)=ncurvs+1
- number of user points in curve i: curveusp(i+1)-curveusp(i)
- user points in curve i are stored in curveusp from position curveusp(i)+1 to curveusp(i+1)

Part KMESH AL Array containing the outer boundary for sets of element groups.
In R^2 the outer boundary consists of curves and array KMESH_al is filled as follows:

Pos. 1 nparts Number of parts the outer boundary consists of

Pos. 2-1+nparts Starting addresses of each part

Per part: (first address: ioutercurvs(1+i) Pos. 1: nouter Number of curves in part
followed by the curve numbers (provided with a sign)
These curve numbers are consecutive, which means the end of one curve is the start of the new one The part itself is closed

In R^3 the outer boundary consists of surfaces and array KMESH_al is filled as follows:

Pos. 1 nparts Number of parts the outer boundary consists of

Pos. 2-1+nparts Starting addresses of each part

Per part: (first address: ioutersurfs(1+i) Pos. 1: nouter Number of surfaces in part followed by the surface numbers (provided with a sign).
The part itself is closed

Part KMESHPAR Array containing information of the global mesh in case of parallel computing. The local array consists of 10 positions with the following contents:

Pos. 1 `npoint_global`, i.e. the number of nodes in the global mesh

Pos. 2 memory management sequence number of array `coor_global`. This array contains the coordinates of the nodes of the global mesh.

Pos. 3 memory management sequence number of array `kmeshl_global`. This array contains the nodal point numbers of the user points of the global mesh.

Pos. 4 memory management sequence number of array `kmeshm_global`. This array contains the nodal point numbers of the curves of the global mesh.

Pos. 5 memory management sequence number of array `kmeshn_global`. This array contains the nodal point numbers of the surfaces of the global mesh.

Pos. 6 memory management sequence number of array `map_global`. This array contains the global nodal point numbers of local nodes.

Pos. 7 memory management sequence number of array `kmeshc_global`. This array contains the nodal point numbers of the elements of the global mesh.

Pos. 8-10

24.3 Array KPROB

Array KPROB contains all information concerning the type of problem to be solved (partial differential equation, boundary conditions, etc.)
A small part of the information is stored directly in array KPROB. This concerns some constants and some small subarrays. Most of the information is stored in the buffer array IBUFFR and can only be reached with the aid of the memory management subroutines. For those arrays the so-called sequence numbers are stored in KPROB.
KPROB consists of two parts.

Part 1: (pos. 1-80) contains special variables and pointers.

Part 2: (pos. 81 - end) contains the sub arrays not stored in IBUFFR.

Usually KPROB refers to 1 problem only. However, if several problems are solved simultaneously (NPROB>1) KPROB consists in fact of a series of NPROB arrays, each of which may be considered as an array KPROB. How to address the correct array KPROB is treated at the end of this section.

Part 1 consists of exactly 80 positions. These positions have the following meaning:

Pos. 1 declared length of array KPROB (to be filled by the user)

Pos. 2 101. This number is used to indicate that this is an array of structure KPROB.

Pos. 3 actual length of array KPROB.

Pos. 4 NUNKP: maximal number of degrees of freedom in nodal points.

Pos. 5 NUSOL: number of degrees of freedom.

Pos. 6 NDTVEC: number of arrays of special structure.

Pos. 7 NTYPE: number of different types to be used for each standard element (including boundary elements).

Pos. 8 NELGRP: number of element groups

Pos. 9 NKBND: number of nodal points with prescribed degrees of freedom.

Pos. 10 NBOUND: number of prescribed degrees of freedom.

Pos. 11 KPROBX: Starting address of KPROB part x in array KPROB. KPROB part x contains information with respect to global unknowns.

Pos. 12 NUMNAT: number of different boundary elements (standard elements).

Pos. 13 NPLTRA: number of points with local transformations.

Pos. 14 KPROBA: Starting address of KPROB part a in array KPROB. Part a contains the number of degrees of freedom in nodal points of standard elements.

Pos. 15 KPROBB: Starting address of KPROB part b in array KPROB. Part b contains the number of elements of the i^{th} standard boundary element.

Pos. 16 KPROBC: Starting address of KPROB part c in array KPROB. Part c contains type numbers of standard elements when NTYPE > 1.

Pos. 17 KPROBD: Starting address of KPROB part d in array KPROB. Part d contains alternative type numbers of standard elements, when NTYPE > 1.

Pos. 18 KPROBE: Memory management sequence number of KPROB part e. Part e contains information of arrays of special structure.

Pos. 19 KPROBF: Memory management sequence number of KPROB part f. Part f contains the number of degrees of freedom in the nodal points.

Pos. 20 KPROBG: Memory management sequence number of KPROB part g. Part g contains information in which nodes essential boundary conditions are given.

Pos. 21 0 This position refers to the old version of KPROB h. This position is not longer in use, however, some subroutines still refer to this position and if the value is not equal to 0, the effect might be incorrect results.

Pos. 22 KPROBI: Memory management sequence number of KPROB part i. Part i contains nodal point numbers of the boundary elements.

Pos. 23 KPROBJ: Starting address of KPROB part j in array KPROB. Part j contains the number of nodal points of each standard boundary element, as well as shape numbers of each boundary element.

Pos. 24 KPROBK: Memory management sequence number of KPROB part k. Part k contains number of degrees of freedom in super elements.

Pos. 25 KPROBL: Memory management sequence number of KPROB part l. Part l contains number of entries in super element matrices.

Pos. 26 KPROBM: Memory management sequence number of KPROB part m. Part m contains the element numbers of the elements in each super element.

Pos. 27 KPROBN: Memory management sequence number of KPROB part n. Part n contains information of the local transforms.

Pos. 28 KPROBO: Starting address of KPROB part o in array KPROB. Part o contains information of boundary conditions of the type u equals constant.

Pos. 29 NSUPER: number of super elements.
When the large matrix can not be kept in-core, it is split into NSUPER parts according to a method of Engelman and Hasbani (1979).

Pos. 30 NBELEM: total number of elements (internal plus boundary elements).

Pos. 31 NDIM: Dimension of the space.

Pos. 32 NBCONS: Number of parts of the boundary where boundary conditions of the type u equals constant are given.

Pos. 33 NPHYS: Number of physical variables in the solution vector (NPHYS $\geq$ NUNKP). In most applications NPHYS = NUNKP. See the Users Manual Section 3.2.2 for an explanation.

Pos. 34 NDEPEN: number of degrees of freedom that depend on other ones. For example in the case of periodical boundary conditions one half of the points with periodical boundary conditions depends on the other half.

Pos. 35 KPROBP: Memory management sequence number of KPROB part p. Part p contains the positions of the degrees of freedom corresponding to each physical quantity.

Pos. 36 KPROBQ: Memory management sequence number of KPROB part q. Part q is reserved for the spectral method.

Pos. 37 KPROBR: Memory management sequence number of KPROB part r. Part r contains the internal renumbering for the matrices.

Pos. 38 KPROBS: Memory management sequence number of KPROB part s. Part s contains information for boundary conditions of the type $\psi\_r = c\_2\psi\_l + c\_1$.

Pos. 39 KPRELGRP: $\sum_{IELGRP=1}^{NELGRP+NUMNAT}$ number of points in standard element IELGRP.

Pos. 40 NPROB: Number of problems that have been stored in KPROB.

Pos. 41 ISIINPUT: Memory management sequence number of array IINPUT. IINPUT contains the numeric translation of the integer input of the problem definition. See Section 4.

Pos. 42 ISRINPUT: Memory management sequence number of array RINPUT. RINPUT contains the numeric translation of the real input of the problem definition. See Section 4.

Pos. 43 KPROBT: Memory management sequence number of KPROB part t. Part t is reserved for non-Newtonian fluids.

Pos. 44 KPROBU: Memory management sequence number of KPROB part u. Part u contains temporary renumbering, used when JMETOD=11,12.

Pos. 45 MAXTRN: First dimension of the array COORTR, containing the real part of KPROB part n.
MAXTRN depends on the type of local transformation.

Pos. 46 KPROBV: Memory management sequence number of KPROB part v. KPROB part v contains information about all contact points.

Pos. 47 KPROBW: Memory management sequence number of KPROB part w. KPROB part w contains information about all contact elements.

Pos. 48 MAXCONTC: This is the largest sequence number of the contact that has been activated.

Pos. 49 ICHANGE_CONTACT: This parameter indicates if new contact points have been created or old contact points have been removed since the last time array KPROB has been updated (1) or not (0).

Pos. 50 KPROBH: Memory management sequence number of KPROB part h. Part h contains new numbers of the degrees of freedom.

Pos. 51 KPROBY: Memory management sequence number of KPROB part y. Part y contains information of the fictitious unknowns.

Pos. 52 KPROBZ: Starting address of KPROB part z in array KPROB.

Pos. 53 KPROBAA: Memory management sequence number of KPROB part aa.

Pos. 54 LAGRANGE_MULTIPLIERS. If 1 this means that the solution vector contains Lagrangian multipliers. The number of vectors of special structure is extended by 1 and this vector corresponds to a solution vector without the Lagrangian multipliers. The output vector written to the file `sepcomp.out` is first transformed to this vector without Lagrangian multipliers before writing.

Pos. 55 MMBLOCKUNKINFO, memory management sequence number for array BLOCKUNKINFO. This array is only used in case of parallel computing.
It contains global block information of the neighboring blocks with respect to the unknowns.
Compare with array BLOCKINFO in KMESH.

Pos. 56 MMSENDINFO, memory management sequence number for array SENDINFO. This array is only used in case of parallel computing.
It contains information of the transport of unknowns between neighboring blocks.
Compare with array TRANSPORTINFO in KMESH.

Pos. 57 MMINNERCONTR, memory management sequence number for array INNERCONTR. This array is only used in case of parallel computing.
It contains information whether an unknown must be taken into account in computing a global inner product.

Pos. 58 MMKPROBAB, memory management sequence number for array KPROB part AB containing information of pair s of periodical points.

Pos. 59 MMKPROBAC, memory management sequence number for array KPROB part AC containing

Pos. 60 MMKPROBAD, memory management sequence number for array KPROB part AD containing the node numbers that have a prescribed cavitation boundary condition.

Pos. 64 nkbndskip, number of prescribed boundary conditions due to skipping of nodes. This means that nkbnd-nkbndskip is the number of nodes with prescribed boundary conditions.

Pos. 65 STKPROBPAR starting address for array KPROBPAR containing extra information in the parallel case.

Pos. 66 MMPETSC, memory management sequence number for array KPROBPETSC containing information for parallel petsc.
In case of a serial computation MMPETSC is set to -1.

Part 2 In this part we describe both the subarrays of KPROB that are stored in KPROB as well as the arrays stored in the buffer array IBUFFR to which KPROB points.

part a contains the number of degrees of freedom in nodal points of standard elements. It is only used when KPROBA > 0.

Length: KPRELGRP (= KPROB(39)).

First position in KPROB: KPROBA

Contents: number of degrees of freedom in nodal points of standard elements
and standard boundary elements sequentially. Hence the number of degrees of freedom in nodal point i of standard element IELGRP is given by:
KPROB (KPROBA - 1 + IKELM + i) with:
IKELM = (IELGRP - 1) × NPELM when KELMA = 0
IKELM = KMESH (KELMA + IELGRP - 1) when KELMA > 0
(NPELM=KMESH(4), KELMA=KMESH(15))
The number of nodes in the boundary elements can be found in KPROB part j.

part b contains the number of elements of the i^{th} standard boundary element. This part is only used when NUMNAT > 0

Length: NUMNAT

First position in KPROB: KPROBB

Contents: number of elements of the i^{th} standard boundary element.

The information of the boundary elements is stored in part i.

Remark

Boundary conditions of the type u equals constant are given by the introduction of extra natural boundary condition elements with type number -2. NBCONS extra element groups are necessary. The last NBCONS element groups are reserved for this purpose.

part c contains type numbers of standard elements.

Length: NELGRP + NUMNAT

First position in KPROB: KPROBC

Contents: type numbers of the standard elements are stored sequentially
(including the boundary elements).
Hence type number of standard element IELGRP is stored in
KPROB (KPROBC - 1 + IELGRP).

part d contains alternative type numbers of standard elements, when NTYPE > 1. This part is only used when NTYPE > 1.

Length: NTYPE × (NELGRP + NUMNAT)

First position in KPROB: KPROBD

Contents: the type numbers of each series of standard elements are stored sequentially.
Hence type number of standard element IELGRP in series number ISERIE is stored in
KPROB (KPROBD - 1 + IELGRP + (ISERIE-1) × (NELGRP + NUMNAT)).

part e contains information of arrays of special structure. This part is only used when NDTVEC > 0.

Length: NDTVEC × (KPRELGRP + 1) (KPRELGRP = KPROB(39)).

First position in KPROB: KPROBE

Contents: Part e can be divided into NDTVEC + 1 parts. The first NDTVEC parts, each of length KPRELGRP refer to the NDTVEC arrays of special structure. In each of these parts the number of degrees of freedom with respect to the vector of special structure in nodal points of standard elements and standard boundary elements is stored sequentially.
Hence the number of unknowns in nodal point i of standard element IELGRP of array of special structure IVEC is given by:
KPROB (KPROBE - 1 + IKELM + i + (IVEC-1)×KPRELGRP) with:
IKELM = (IELGRP - 1) × NPELM when KELMA = 0
IKELM = KMESH (KELMA + IELGRP - 1) when KELMA > 0

The last part of length NDTVEC contains information of the number of degrees of freedom in the actual nodal points with respect to the $IVEC^{th}$ array of special structure.

When the number of degrees of freedom in the nodal points is constant, then **minus** this number is stored in KPROB (KPROBE - 1 + KPRELGRP × NDTVEC + IVEC), otherwise (number of degrees of freedom not the same in each nodal point), information of the number of degrees of freedom in each nodal point is stored in array IBUFFR.
In that case KPROB (KPROBE - 1 + KPRELGRP × NDTVEC + IVEC) contains the memory management sequence number of array KPROBE_IVEC containing the number of degrees of freedom per point.

Array KPROBE_IVEC contains the number of degrees of freedom in each nodal point is stored accumulated. Hence:
KPROBE_IVEC (IPKPRE) = 0
number of degrees of freedom in nodal point INODP =
KPROBE_IVEC (IPKPRE + INODP) - KPROBE_IVEC (IPKPRE + INODP - 1)

part f contains the number of degrees of freedom in the nodal points. This part is only used when KPROBA > 0, and hence KPROBF > 0 and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KPROBF as sequence number.

Length: NPOINT + 1 positions.

Contents number of degrees of freedom (prescribed or not), in each nodal point. The number of degrees of freedom are accumulated. Hence, let KPROBf be the array stored in IBUFFR, then:
KPROBf(1) = 0
number of degrees of freedom in nodal point INODP =
KPROBf(1+INODP) - KPROBf(INODP)

part g contains information in which nodes essential boundary conditions are given. This part is only used if essential boundary conditions are present (KPROBG>0) and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KPROBG as sequence number.

Length: 2 × NKBND

Contents locally array KPROB_part_g may be regarded as a two-dimensional array of size 2 × NKBND.
KPROB_part_g(1,i) contains the i^{th} nodal point sequence number in which essential

boundary conditions are given.
KPROB_part_g(2,i) contains a bit pattern indicating which degrees of freedom are prescribed. The bit pattern is counted from right to left and each bit corresponds to a degree of freedom. Bit 1 means degree of freedom is prescribed, bit 0 means it is not prescribed. Hence if for example the degrees of freedom 1 2 and 4 are prescribed, then the corresponding bit pattern is 1011 and the corresponding number stored in KPROB_part_g(2,i) is equal to 11 ($2^3 + 2^1 + 2^0$).

part h contains new numbers of the degrees of freedom. This part is only used when NBOUND > 0 or KELMJ > 0. and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KPROBH as sequence number.

Length: NUSOL

Contents The degrees of freedom are renumbered, since the nodal point numbers may be renumbered (KELMJ > 0) and/or since the prescribed degrees of freedom are put at the rear (NBOUND > 0). The new positions of the degrees of freedom are stored in KPROB part h sequentially.
KPROB part h is organized such that each unknown refers to exactly one position in the solution array, whereas KPROB part r may contain double references.
In fact internally the unknowns in the solution array are stored in the sequence defined by the node numbers. The numbering defined by KPROB part h is used to store the matrices. Hence matrices and solution vector have a different sequence. Subroutines manipulating with matrices, map the solution vector onto a vector with the same sequence as the matrices.

part i contains nodal point numbers of the boundary elements. This part is only used when NUMNAT > 0, and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KPROBI as sequence number.

Length: $\sum_{boundary\,elements}$ number of nodal points in boundary element.

Contents: nodal point numbers of each boundary element sequentially.
The elements must be numbered according to increasing standard boundary element number. Compare with KMESH part c.

part j contains the number of nodal points of each standard boundary element, as well as shape numbers of each boundary element. This part is only used when KPROBJ > 0.

Length: NUMNAT$*2$

First position in KPROB: KPROBJ

Contents: the number of nodal points of each standard boundary element, sequentially, followed by the shape numbers of each standard boundary element. Hence INPELM = KPROB (KPROBJ - 1 + IBOUN) gives the number of nodal points in standard boundary element IBOUN.
ISHAPE = KPROB (KPROBJ - 1 + IBOUN + NUMNAT) gives the shape number of standard boundary element IBOUN. For a definition of the shape numbers, see KMESH part f.

part k contains number of degrees of freedom in super elements. This part is only used when NSUPER > 0, and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KPROBK as sequence number.

Length: NSUPER

Contents: number of degrees of freedom in each super element are stored accumulated.

Hence: number of degrees of freedom in super element 1 = KPROB k (1).
number of degrees of freedom in super element i = KPROB k (i) - KPROB k ($i-1$).

part l contains number of entries in super element matrices. This part is only used when NSUPER > 0. and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KPROBL as sequence number.

Length: NSUPER

Contents: number of entries in super element matrices.

part m contains the element numbers of the elements in each super element. This part is only used when NSUPER > 0, and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KPROBM as sequence number.

Length: 2 × NSUPER

Contents: KPROB part m contains information of the element numbers of the elements per super element.

The actual information is stored in part $m1$, which is always kept on file 3.
KPROB m (2×ISUPER-1) contains the length of part $m1$ corresponding to super element ISUPER.
KPROB m (2×ISUPER) contains the record number of part $m1$
corresponding to super element ISUPER.

KPROB part $m1$ is always stored on backing storage (file 3).
For each super element it contains the following data:
Pos. 1 - NELGRP+NUMNAT: Number of elements of each element group with respect to super element ISUPER.
Pos. NELGRP+NUMNAT+1, . . . : Element numbers of elements in the super element, stored for increasing element group number.

part n contains information of the local transforms. This part is only used when NPLTRA > 0, and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KPROBN as sequence number.

Length: NPLTRA × (1 + NDIM + MAXTRN × INTLEN)

Contents: KPROB part n contains information of the local transformations.
The actual information is stored in 2 parts.

To compute the starting addresses of these parts in IBUFFR/BUFFER, the following statements must be used:

```
ipkprb = iniprb ( iprob, kprob )
ipkprobn = iniget ( kprob(ipkprb+27) )
ipcoortr = inidbl(ipkprobn+ndim*npltra+1)

call subroutxxx ( ibuffr(ipkprobn), buffer(ipcoortr), ... )
```

and the corresponding subroutine starts with:

```
subroutxxx ( kprobn, coortr, ... )
```

Meaning of these parameters:

IPKPRB last address of part of KPROB corresponding to IPROB-1.
If IPROB=1, ipkprb=0

IPKPROBN Starting address of integer part of KPROB part n.

IPCOORTR Starting address of real part of KPROB part n (COORTR).
Mark that there is a plus 1 in the address. This extra position is meant to be sure

that the first position of the real part does not overwrite the last position of the integer part. This might be the case for NPLTRA and NDIM both odd.

INIPRB Computes starting address of part of KPROB corresponding to specific problem number, See end of this section.

INIGET gets starting address of integer array in IBUFFR. See 22.6.16.

INIDBL Transforms a starting address in IBUFFR into a starting address in BUFFER. See 22.6.22.

The contents of these parts are given by:

part 1: (length NDIM × NPLTRA) contains the sequence numbers of the degrees of freedom corresponding to the points with local transforms with respect to the solution vector. These sequence numbers are stored sequentially: NDIM numbers for each node (2D only).

part 2: (length MAXTRN × NPLTRA reals) contains the real information, depending on the the parameter MAXTRN.
This part may be considered as an array COORTR(1:MAXTRN,1:NPLTRA).
If MAXTRN = 2, the standard transformation is applied.
COORTR(1,i) contains n_x and COORTR(2,i) contains n_y, i.e. the components of the normal vector.
The transformation matrix is defined as:

$$\mathbf{R} = \begin{bmatrix} n_x & -n_y \\ n_y & n_x \end{bmatrix} \tag{24.3.1}$$

If MAXTRN = 4, the symmetric transformation is applied. This is also the case if the transformation $(-u_n, u_t)$ is used.
COORTR(1,i) contains R_{11}, COORTR(2,i) contains R_{12} and so on. The transformation matrix R is defined by

$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix} \tag{24.3.2}$$

The back transformation is given by

$$u_{old} = R^{-1} u_{transformed} \tag{24.3.3}$$

If MAXTRN = 8, the non-symmetric transformation is applied.
COORTR(1,i) contains R_{11}, COORTR(2,i) contains R_{12} and so on. COORTR(5,i) contains V_{11}, COORTR(6,i) contains V_{12} and so on. The transformation matrices R and V are defined by

$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix} \qquad \mathbf{V} = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix} \tag{24.3.4}$$

The back transformation is the same as for MAXTRN=4.

In the case of NDIM = 3, the standard transformation matrix **R** is given by

$$\mathbf{R} = \begin{bmatrix} n_x & t\_1_x & t\_2_x \\ n_y & t\_1_y & t\_2_y \\ n_z & t\_1_z & t\_2_z \end{bmatrix} \tag{24.3.5}$$

with **n** the normal and **t_1** and **t_2** the two tangential vectors.

part o contains information of boundary conditions of the type u equals constant. This part is only used when NBCONS > 0.

Length: NBCONS

First position in KPROB: KPROBO

Contents: To describe boundary conditions of the type u equals constant, extra natural boundary condition elements with type number -2 are introduced. Along each part of the boundary where boundary conditions of the type u equals constant is given, we introduce a new element group. Hence NBCONS extra boundary elements are necessary. Furthermore the degree of freedom that is constant for the boundary element group NUMNAT - NBCONS + i is stored in KPROB (KPROBO + i - 1).

part p contains the positions of the degrees of freedom corresponding to each physical quantity. This part is only used when NUNKP > 1 or NPHYS > 1 or KPROBA > 0, and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KPROBP as sequence number.

Length: NPOINT × NPHYS

Contents: KPROB part p contains NPHYS arrays of length NPOINT each. The i^{th} array corresponds to the i^{th} physical variable in nodal point j in the solution array. When the physical variable i is not available in point j than a zero is stored.

part r contains the internal renumbering for the matrices. This part is only used when NBOUND > 0 or NDEPEN > 0 or KELMJ > 0, and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KPROBR as sequence number.

Length: NUSOL

Contents: KPROB part r contains the internal numbering with respect to the building of matrices. In general KPROB part h and KPROB part r are identical arrays (in fact they refer to the same array), however, if NDEPEN > 0, there is an essential difference. KPROB part h is organized such that each unknown refers to exactly one position in the solution array, whereas KPROB part r may contain double references. For example in the case of periodical boundary conditions, degrees of freedom in connected points refer to the same unknown, thus reducing the number of degrees of freedom.

part s contains information for boundary conditions of the type $\psi\_r = c\_2\psi\_l + c\_1$. This part is only used when NDEPEN > 0, and is stored in the buffer array IBUFFR. Its starting address can be found with the memory management subroutines using KPROBS as sequence number.

Length: NDEPEN integers plus 2 ∗ NDEPEN reals.

Contents: KPROB part s contains information of boundary conditions of the type $\psi\_r = c\_2\psi\_l + c\_1$.
The first NDEPEN reals are used to store the constants c_1 per degree of freedom that must be added for the boundary condition. For periodical boundary conditions c is equal to zero. The second NDEPEN reals are used to store the constants c_2 in the same sequence.
The second part of KPROB part s (NDEPEN integers) contains references of the degrees of freedom NUSOL-NBOUND-NDEPEN+1 to NUSOL-NBOUND (ψ_r) in array USOL to the corresponding degrees of freedom ψ_l in the first NUSOL-NBOUND-NDEPEN positions of USOL.

part u Temporary renumbering, used when JMETOD = 11,12.

part v KPROB part v contains information about the contact points.
This part is only available if KPROB(46)>0.
As soon as a call to a contact subroutine is executed this array is either created or activated. It is not created by one of the starting subroutines. The fact that KPROB part v exists does not imply that there are any contact points.
KPROB part v consists of two sub parts KPROBV_part_1 and KPROBV_part_2, which are stored as one array.

Length:

KPROBV_part_1 has length 3 × NSURFS (number of surfaces).
KPROBV_part_2 has length NPOINT (number of nodal points).
Contents:

Locally KPROBV_part_1 may be considered as a two-dimensional array of size (1:3,1:NSURFS).

Pos. (1,i) contains the number of points that make contact in the i^{th} contact surface.

Pos. (2,i) contains the number of points that make no contact in the i^{th} contact surface. Hence the sum of (1,i) and (2,i) is the total number of points in the surface.

Pos. (3,i) contains the starting address of contact points in the i^{th} contact surface in part 2. The starting address is relative to part 2, which means that the lowest number is equal to 1.

Part 2 contains for each contact surface all nodal point numbers in the sequence defined by KPROBV_part_1(3,i). The sequence for each contact surface is: first all points that make contact, and then the rest of the points i.e. the points that make no contact.

part w KPROB part w contains information about the contact elements.
This part is only available if KPROB(46)>0 and hence KPROB(47)>0.
As soon as a call to a contact subroutine is executed this array is either created or activated. It is not created by one of the starting subroutines. The fact that KPROB part w exists does not imply that there are any contact points.
KPROB part w consists of two sub parts KPROBW_part_1 and KPROBW_part_2, which are stored as one array.

Length:

KPROBW_part_1 has length 4 × NSURFS (number of surfaces).
KPROBW_part_2 has length 2 × NSURFS plus the length of KMESH part n.
Contents:

Locally KPROBW_part_1 may be considered as a two-dimensional array of size (1:4,1:NSURFS).

Pos. (1,i) contains the number of elements that make contact in the i^{th} contact surface. An element makes contact if all nodes in the element make contact.

Pos. (2,i) contains the number of elements that make no contact in the i^{th} contact surface. Hence the sum of (1,i) and (2,i) is the total number of elements in the surface.

Pos. (3,i) contains the surface number of the i^{th} contact surface.

Pos. (4,i) contains the starting address of contact elements in the i^{th} contact surface in part 2. The starting address is relative to part 2, which means that the lowest number is equal to 1.

Part 2 contains for each contact surface all element numbers in the sequence defined by KPROBW_part_1(4,i). The sequence for each contact surface is: first all elements that make contact, and then the rest of the elements i.e. the elements that make no contact.

part x Part x is used to store information about the global unknowns. The global unknowns get always numbered behind the normal unknowns, in order to minimize the profile.
Part x has the following contents:

Pos. 1 NGLGRP: number of element groups with respect to the global unknowns.

Pos. 2 NGLUNK: total number of global unknowns.

Pos. 3 NGLCOUPL: number of global unknowns that must be used for global renumbering

These positions are followed by 6 × NGLGRP positions each referring to one element group. If we consider this as a 6 × NGLGRP array then this array has the following contents:

Pos. (1,i) ITYPE, defines the type number of the corresponding elements

Pos. (2,i) INUNKPGR, number of degrees of freedom for the specific element group (usually 1).

Pos. (3,i) ITYPREGION Defines the type of region on which the global unknown is defined. Possible values:

1. The region consists of a series of curves
2. The region consists of a series of surfaces
3. The region consists of a series of volumes
4. The complete region is used

Pos. (4,i) ISHAPE, shape number of the elements according to Table 2.2.1 in the Users Manual Section 2.2.
This shape number is only used in case of curve elements.

Pos. (5,i) ICRV1 First curve (surface/volume) number.

Pos. (6,i) ICRV2 Last curve (surface/volume) number.

part y Part y is used to store information about the fictitious unknowns. KPROBy is filled as follows:

Pos. 1 nfictgrp (number of fictitious groups)

Pos. 2 nfictunk total number of lambdas

Pos. 3 nfictpnt total number of nodes with lambdas

Pos. 4 memory management sequence number of kprobfict

Pos. 5-10 not yet used

Pos. 11-11+nfictgrp starting addresses with information per group The last one refers to the end position plus one

Per group the following information is stored:

1 itype type number of element

2 inunkpgr number of degrees of freedom per lambda point

3 iregionlam type of region where the lambda's are defined: Possible values

   1 curves

   2 surfaces

4 ishape Shape of lambda basis functions

5 icurve1 First curve of surface

6 icurve2 Last curve of surface

7 istructgrp Structural element group

8 ifluidgrp1 First fluid element group

9 ifluidgrp2 Last fluid element group

10 inunkp_elm Number of lambda's in element

11 nelem Number of elements for this group

12 ishape_curve Shape number of "curve" elements

Array KPROBFICT is stored in the buffer array IBUFFR. The length of this array is 2 × nfictpnt and it may be considered as a two dimensional array of size KPROBFICT(1:2,1:nfictpnt). KPROBFICT(1,i) contains the structural element corresponding to the i^{th} fictitious node. KPROBFICT(2,i) contains the fluid element corresponding to the i^{th} fictitious node.

part z KPROB z is only available if KPROB(52) > 0.
In that case KPROB part z has length NELGRP+NUMNAT.
The first NELGRP positions refer to the element groups and the last NUMNAT positions to the boundary groups.
For each (boundary) element group either a zero or a memory management sequence number is stored.
A zero means that no extra information for that element group is available.
If a memory management sequence number is stored, this refers to an extra array corresponding to the element group. This extra array is dependent on the type of elements.
For example in case of boundary elements defined by a cross-section of a fixed grid with an obstacle this extra array contains the relative position of the cross section point in the element.

part aa KPROB aa is only available if KPROB(53) > 0.
This array of length NELEM is stored in IBUFFR with memory management sequence number KPROB(53).
KPROBaa(ielem) = 1, means that element ielem must be skipped when creating the large matrix,
KPROBaa(ielem) = 0, means that it must not be skipped.

Array BLOCKUNKINFO This array is only used in case of parallel computing.
It is stored in array IBUFFR with memory management sequence number KPROB(55).
It contains the global block information for block number IBLOCK. In this case it is related to non-prescribed degrees of freedom and not to nodes.
Contents:

Pos. 1 m (number of neighbor blocks)

Pos. 2i block number of i-th neighbor

Pos. 2i+1 starting address of i-th neighbor in array SENDINFO.

Pos. 2m+i+1 starting address of extra information of i-th neighbor in array BLOCKUNKINFO. This starting address is at least equal to 3m+2.

Pos. istart where istart is the starting address referred to in 2m+i-1 is the first position with extra information. This extra information starts with the number of neighbor blocks m_i of block i that are also neighbor of the present block. This is followed by 2 m_i positions containing the sequence number of that neighbor block followed by a starting address in array sendinfo.

Array SENDINFO This array is only used in case of parallel computing.
It is stored in array IBUFFR with memory management sequence number KPROB(56).
It contains information of the transport of unknowns between neighboring blocks.
The starting address of each neighbor block is stored in array BLOCKUNKINFO.
For each neighbor block the following information is stored:

Pos. 1 number of unknowns (degrees of freedom) n on common side of both blocks

Pos. 2 ... n+1 local unknowns numbers in neighbor block i

In the second part information of neighbors of neighbors as mentioned in BLOCKUNKINFO is stored. The starting address can be found in BLOCKUNKINFO
For each neighbor block the following information is stored:

Pos. 1 number of unknowns (degrees of freedom) n_{ij} in the points on the interface $i-j$ that are also neighbor of the present block.

Pos. 2 ... n_{ij}+1 local sequence numbers of unknowns corresponding to nodes in neighbor block i

Array INNERCONTR This array is only used in case of parallel computing.
INNERCONTR consists of 2 parts INNERCONTR_ORIG and NUMBLOCKS. This array is

stored in array IBUFFR with memory management sequence number KPROB(57).
Part 1 (INNERCONTR_ORIG) has length NRUSOL
It indicates if an unknown in the block must be used in a global inner product or not.
innercontr(i) = 1: take unknown i into account.
innercontr(i) = 0: take unknown i not into account.
The unknown in the block with smallest number is taken into account
Part 2 (NUMBLOCKS) contains for each boundary unknown the number of blocks it is lying in. This part is organized as follows:
Pos. 1 contains the number of unknowns (m) of the block that are also part of another block.
The next $2 \times m$ positions can be considered as a two-dimensional array inumblocks(2,m) with
inumblocks(1,i) the sequence number of the unknown at the interface and
inumblocks(2,i) the number of blocks this unknown is lying in (at least 2).

So part 2 of this array can be used as follows:

```
m = innercontr(nrusol+1)
call subrout ( m, innercontr(nrusol+2), ... )
  .
  .
  .
end

subroutine subrout ( m, innercontr(nrusol+2), ... )
integer m, inumblocks(2,m)
  .
  .
  .
end
```

Array KPROBAB This array is only used in case of periodical boundary conditions.
It is stored in array IBUFFR with memory management sequence number KPROB(58).
KPROB part ab contains "pairs" of periodical degrees of freedom.
KPROBAB is filled as follows:

Pos. 1: NPAIR: number of pairs

Pos. 2 ... NPAIR+1: starting addresses for each pair.

For each pair from starting address:

Pos. 1 number of degrees of freedom in pair, followed by sequence numbers of these degrees of freedom

Array KPROBPAR This array of length 10 with starting address KPROB(65) contains the following extra information for parallel computing:

Pos. 1: nusol_global, i.e. global number of degrees of freedom

Pos. 2: nrusol_global, i.e. global number of free degrees of freedom

Pos. 3: memory management sequence number of kprobf_global (Not yet available)

Pos. 4: memory management sequence number of kprobh_global kprobh_global contains the mapping from the global degrees of freedom to the local ones per node. This array is only available at node 1. The Positions 1 to iacnodes (number of parallel nodes) contains the starting positions of each node (including node 1).
Each starting position contains the number of free degrees of freedom in the node. It is followed by the global sequence numbers of the local free degrees of freedom.

Pos. 5-10: -

Remark: If KPROB contains more than one problem then it may be necessary to find the starting address of the actual KPROB array. In order to find this address, function subroutine INIPRB may be used. This subroutine may be called as follows:

```
ipkprb = iniprb ( iprob, kprob )
```

IPROB indicates the problem number and IPKPRB the starting address of the actual array KPROB corresponding to IPROB minus one. Hence iniprb(1,kprob) returns with the value 0, and in order to get for example nunkp for problem iprob we can use the following statements.

```
ipkprb = iniprb ( iprob, kprob )
nunkp = kprob(ipkprb+4)
```

24.4 Array ISOL

Array ISOL contains some information about the solution array. It concerns information about the structure as well as a memory management sequence number referring to the actual solution array. Array ISOL itself consists of 5 positions only.
These positions are filled as follows:

Pos. 1 Memory management sequence number referring to the actual solution array.

Pos. 2 110. This number is used to indicate that this is an array of the structure ISOL.

Pos. 3 ITYPE, indicating the type of co-ordinate system as well as if the array is real or complex. ITYPE consists of two parts IREAL and ICOORSYSTEM according to:
ITYPE = IREAL + 10 × (ICOORSYSTEM-1), with

IREAL indicates whether the array is real (0) or complex (1)

ICOORSYSTEM defines the type of co-ordinate system corresponding to the solution vector. Possible values:

1 Cartesian co-ordinate system
2 Axi-symmetric co-ordinate system
3 Polar co-ordinate system

Pos. 4 contains 1000 × (IPROB-1), where IPROB denotes the problem sequence number to which ISOL corresponds.

Pos. 5 NUSOL, i.e. the number of degrees of freedom stored in the solution vector. This number stands for all degrees of freedom, whether they are prescribed or not.

The solution vector USOL

The solution vector is stored in the buffer array BUFFER and will be referred to as USOL. In order to address this array it is necessary to use the memory management subroutines of Section 22.

Length: NUSOL double precisions or NUSOL complex double precisions depending on ISOL(3).

Contents: The solution vector contains all degrees of freedom. If no renumbering of unknowns is applied, i.e. if KPROB(21) = 0, the sequence is: first all degrees of freedom in node 1, then in node 2 and so on.
If KPROB(21) > 0, a renumbering is applied. This is done because the nodal points are renumbered, or because there are essential boundary conditions which are put at the end of the array, or because there are periodical boundary conditions.
In that case the sequence of KPROB part h is used.

In order to find the i^{th} degree of freedom of node j in the solution vector, it is necessary to apply the memory management subroutines. The following piece of code performs this task.

```
      program example
      implicit none
      integer isol(5), kprob(100)
      integer indprf, indprh, nunkp, ipkprf, ipkprh,
     +        degree_of_freedom_i, point_j
      double precision value

c     --- Local variables
```

```
c     degree_of_freedom_i  Degree of freedom required
c     indprf    If 0 KPROB part f has not been filled, otherwise it indicates
c               the memory management sequence number corresponding to KPROB f
c     indprh    If 0 KPROB part h has not been filled, otherwise it indicates
c               the memory management sequence number corresponding to KPROB h
c     ipkprf    Starting address of array KPROBf in IBUFFR
c     ipkprh    Starting address of array KPROBh in IBUFFR
c     ipusol    Starting address of array USOL in BUFFER
c     isol      Standard solution information array
c     kprob     Standard problem information array
c     nunkp     maximum number of unknowns per point
c     point_j   Nodal point number required
c     value     Computed value
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

c     --- Common blocks

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

c
c                 Blank common
c    Working storage for all kinds of operations
c
c    ibuffr  i/o  General SEPRAN buffer array
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

c    --- Subroutines called:

      integer iniget, inidgt
      double precision value, getsolution

c     getsolution  Example function subroutine to get the value of
c                  a component of the solution in a point
c     ini070       Activate array in BUFFER or IBUFFR
c     inidgt       Compute pointer in BUFFER
c     iniget       Compute pointer in IBUFFR
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

c     --- Statements to start SEPRAN, read the mesh, make computations
c         and so on


c     --- activate the necessary arrays

      indprf = kprob(19)
      indprh = kprob(21)
      if ( indprf.gt.0 ) call ini070(indprf)
      if ( indprh.gt.0 ) call ini070(indprh)
      call ini070(isol(1))

c     --- Compute pointers of arrays
```

```
      if ( indprf.eq.0 ) then

c     --- Number of unknowns per point is constant

         ipkprf = 1

      else

c     --- Number of unknowns per point is constant

         ipkprf = iniget(indprf)

      end if

      if ( indprh.eq.0 ) then

c     --- No renumbering of degrees of freedom

         ipkprh = 1

      else

c     --- Degrees of freedom are renumbered

         ipkprh = iniget(indprh)

      end if
      ipusol = inidgt(isol(1))
      nunkp = kprob(4)

      value = getsolution ( degree_of_freedom_i, point_j,
     +                      buffer(ipusol), ibuffr(ipkprf),
     +                      ibuffr(ipkprh), indprf, indprh, nunkp )

c     --- Rest of statements

      end

c     --- Actual manipulation function

      function getsolution ( degree_of_freedom_i, point_j, USOL,
     +                       kprobf, kprobh, indprf, indprh, nunkp )
      implicit none
      integer degree_of_freedom_i, point_j,
     +        kprobf(*), kprobh(*), kprob(*)
      double precision USOL(*), getsolution

c     --- Local parameters
c
c    ipointer  refers to position in USOL where node can be found
c
      integer ipointer
      if ( indprf.eq.0 ) then

c     --- Number of unknowns per point is constant
```

```
         ipointer = (point_j-1)*nunkp+degree_of_freedom_i

      else

c     --- Number of unknowns per point is not constant, use KPROB part f

         ipointer = kprobf(point_j)+degree_of_freedom_i

      end if
      if ( indprh.gt.0 ) then

c     --- Degrees of freedom have been renumbered

         ipointer = kprobh(ipointer)

      end if

      getsolution = USOL(pointer)

      end
```

24.5 Array IRHSD

Array IRHSD contains some information about the right-hand side vector. It concerns information about the structure as well as a memory management sequence number referring to the actual right-hand side.
Array IRHSD itself consists of 5 positions only.
These positions are filled as follows:

Pos. 1 Memory management sequence number referring to the actual right-hand side.

Pos. 2 110. This number is used to indicate that this is an array of the structure IRHSD.

Pos. 3 ITYPE, indicating the type of co-ordinate system as well as if the array is real or complex. ITYPE consists of two parts IREAL and ICOORSYSTEM according to:
ITYPE = IREAL + 10 × (ICOORSYSTEM-1), with

IREAL indicates whether the array is real (0) or complex (1)

ICOORSYSTEM defines the type of co-ordinate system corresponding to the solution vector. Possible values:

1 Cartesian co-ordinate system

2 Axi-symmetric co-ordinate system

3 Polar co-ordinate system

Pos. 4 contains 1000 × (IPROB-1), where IPROB denotes the problem sequence number to which IRHSD corresponds.

Pos. 5 NUSOL, i.e. the number of degrees of freedom stored in the solution vector. This number stands for all degrees of freedom, whether they are prescribed or not.

The vector RHSD containing the right-hand side has exactly the same structure as the solution vector treated in Section 24.4.
In general the last positions of RHSD refer to prescribed degrees of freedom. For that reason they are not important and usually set to 0.

24.6 Array INTMAT

Contains integer information concerning the structure of the large matrix.
INTMAT consists of 5 positions:

Pos. 1 JMETOD + 100 × (INVMAT + 2 × IBCMAT) + 1000 × (IPROB-1), where JMETOD, IBCMAT and INVMAT define the storage scheme as described in the Users Manual Section 3.2.4.
IPROB denotes the problem number INTMAT corresponds to.

Pos. 2 102

Pos. 3 INTM1 Memory management sequence number of the first part of the actual information.

Pos. 4 INTM2 Memory management sequence number of the second part of the actual information.

Pos. 5 KAMAT | KAMAT | gives the minimum length required in-core to store the large matrix, or parts of it when it is partitioned, in reals. When KAMAT > 0, the large matrix is kept in-core entirely, when KAMAT < 0, it is partitioned.

Actual information

The actual information is divided into two parts: part 1 is connected with the non-prescribed degrees of freedom, part 2 gives the connections between prescribed and non-prescribed degrees of freedom.

Part 1 is stored in array IBUFFR. Its starting address can be found by the memory management subroutines using INTMAT(3) as sequence number.

Contents The contents of part 1 depend on the parameter JMETOD.

See also array MATR (24.7).

JMETOD = 1, 2, 3, 4, 19 (direct solution method)

At this moment a profile (skyline method, wave front method) is chosen.

Length: 2 × (NUSOL - NBOUND)

Part 1 is divided into two parts each of length NUSOL - NBOUND

Part 1^a (First NUSOL - NBOUND positions) contains the indices of the diagonal elements s_{ii} in the large matrix, corresponding to the i^{th} row.
Hence let the large matrix be stored in an array called AMAT (part of array IBUFFR), then:
s_{ii} = AMAT (INTMAT $part1^a$ (i), $i = 1$ (1) NUSOL - NBOUND)

Remark: AMAT(1) is supposed to contain the first element of the large matrix; the numbering of KPROB part h is used when KPROBH > 0.

Part 1^b (Next NUSOL - NBOUND positions)
The column numbers j of the at most left elements of the profile for the different rows are stored. Hence INTMAT $part1^b$ (i) contains the smallest column number j such that $s_{ij} \neq 0$.

Example 24.6.1

Consider the following (8 × 8) matrix S.

$$S = \begin{bmatrix} S_{11} & S_{12} & 0 & S_{14} & 0 & 0 & 0 & 0 \\ S_{21} & S_{22} & 0 & S_{24} & 0 & 0 & 0 & 0 \\ 0 & 0 & S_{33} & S_{34} & 0 & 0 & 0 & 0 \\ S_{41} & S_{42} & S_{43} & S_{44} & S_{45} & S_{46} & S_{47} & 0 \\ 0 & 0 & 0 & S_{54} & S_{55} & S_{56} & S_{57} & 0 \\ 0 & 0 & 0 & S_{64} & S_{65} & S_{66} & S_{67} & 0 \\ 0 & 0 & 0 & S_{74} & S_{75} & S_{76} & S_{77} & S_{78} \\ 0 & 0 & 0 & 0 & 0 & 0 & S_{87} & S_{88} \end{bmatrix}$$

The lower triangle is stored row after row, the upper triangle column after column. Hence when the matrix is non-symmetrical (JMETOD = 2, 4), the matrix is stored in the following way:

$S_{11}, S_{21}, S_{22}, S_{12}, S_{33}, S_{41}, S_{42}, S_{43}, S_{44}, S_{34}, S_{24}, S_{14},$
$S_{54}, S_{55}, S_{45}, S_{64}, S_{65}, S_{66}, S_{56}, S_{46}, S_{74}, S_{75}, S_{76}, S_{77},$
$S_{67}, S_{57}, S_{47}, S_{87}, S_{88}, S_{78}$

For a symmetrical or hermitian matrix (JMETOD = 1, 3, 19) the storage is:

$S_{11}, S_{21}, S_{22}, S_{33}, S_{41}, S_{42}, S_{43}, S_{44}, S_{54}, S_{55}, S_{64}, S_{65}, S_{66},$
$S_{74}, S_{75}, S_{76}, S_{77}, S_{87}, S_{88}$

INTMAT $part1^a$ becomes:

1, 3, 5, 9, 14, 18, 24, 29 (non-symmetrical storage)

1, 3, 4, 8, 10, 13, 17, 19 (symmetrical storage)

INTMAT $part1^b$ becomes:

1, 1, 3, 1, 4, 4, 4, 7

The relation between the matrix S and the storage in array AMAT (part of array IBUFFR), is given by:

$s_{ij} = 0$ when ij and $j <$ INTMAT 1^b (i)
$s_{ij} = 0$ when ij and $i <$ INTMAT 1^b (j)

s_{ij} = AMAT (INTMAT $1^a(i) - i + j)$ when $i \geq j$
s_{ij} = AMAT (INTMAT $1^a(j) - i + j)$ when ij and matrix is
non-symmetrical.

Remark: The profile of the matrix is always symmetrical.

JMETOD = 5, 6, 7, 8 (compact symmetrical storage scheme)

Length: NUSOL-NBOUND-NDEPEN + 1 + LBLOCKS + $\sum_{i=1}^{NUSOL-NBOUND-NDEPEN}$ number of free degrees of freedom connected with i.

Part 1 is divided into three parts:

Part 1^a First NUSOL-NBOUND-NDEPEN + 1 positions, contains the number of degrees of freedom with unknown number smaller than i connected with degree of freedom i (accumulated).

Part 1^b Next positions, contains the unknown numbers of these degrees of freedom. These unknowns are sorted in increasing order for each row. The length of this part is given by INTMAT1a(NUSOL-NBOUND-NDEPEN+1)

Part 1^c Next positions, contains information of the blocks with respect to unknowns per

point.
A block is defined as a set of free degrees of freedom in a nodal point. Hence even if we have a constant number of unknowns per point, still the size of the various blocks may be different since the number of prescribed boundary conditions may differ per point.
LBLOCKS is equal to NBLOCKS+2, where NBLOCKS is the number of blocks. NBLOCKS is at most equal to NPOINT, but may be smaller, if the number of free degrees of freedom in a point is zero. This may be for example the case due to essential boundary conditions.
Pos. 1 of part 1^c contains the parameter NBLOCKS, the positions 2 to LBLOCKS contain the size of the blocks accumulated.
INTMAT1c(2) = 0
size of block i is INTMAT1c(2+i)-INTMAT1c(1+i).

Example 24.6.2

Let the following degrees of freedom be connected to the degrees of freedom given:

1:	3	6	2	5	
2:	1	3	8	7	
3:	7	2	1	4	5
4:	3	6	7		
5:	3	1	8		
6:	4	1	7	8	
7:	2	3	4	6	
8:	6	5	2		

Then part 1^a and part 1^b contain:

	Part 1^a	Part 1^b			
1:	0	-			
2:	0	1			
3:	1	1	2		
4:	3	3			
5:	4	1	3		
6:	6	1	4		
7:	8	2	3	4	6
8:	12	2	5	6	
	15				

or in the one-dimensional storage:
Part 1^a: 0 0 1 3 4 6 8 12 15
Part 1^b: 1 1 2 3 1 3 1 4 2 3 4 6 2 5 6

The column numbers corresponding to row i can be found in the following way:

number of columns = Part 1^a (i+1) - Part 1^a (i)
column numbers are stored in part 1^b from position Part 1^a (i) + 1.
The large matrix is stored as follows:

For a symmetrical matrix (JMETOD = 5, 7) first the diagonal is stored and then the lower triangular part in the same sequence as INTMAT part 1^b. Hence for the example:

$s_{11}, s_{22}, s_{33}, s_{44}, s_{55}, s_{66}, s_{77}, s_{88}$
$s_{21}, s_{31}, s_{32}, s_{43}, s_{51}, s_{53}, s_{61}, s_{64}, s_{72}, s_{73}, s_{74}, s_{76}, s_{82}, s_{85}, s_{86}$

For a non-symmetrical matrix (JMETOD = 6, 8) first the diagonal is stored then the lower triangular part and finally the upper triangular part in the same sequence as

INTMAT part 1^b. Hence for the example:

$s^{11}, s^{22}, s^{33}, s^{44}, s^{55}, s^{66}, s^{77}, s^{88}$

$s_{21}, s_{31}, s_{32}, s_{43}, s_{51}, s_{53}, s_{61}, s_{64}, s_{72}, s_{73}, s_{74}, s_{76}, s_{82}, s_{85}, s_{86}$

$s_{12}, s_{13}, s_{23}, s_{34}, s_{15}, s_{35}, s_{16}, s_{46}, s_{27}, s_{37}, s_{47}, s_{67}, s_{28}, s_{58}, s_{68}$

JMETOD = 9, 10 (compact storage scheme)

See JMETOD = 6, 8, however, for each row all column numbers are stored, not only those with column numbers smaller than i.

Example 24.6.3

See example 24.6.2.

Part 1^a and part 1^b contain:

	Part 1^a	Part 1^b				
1:	0	2	3	5	6	
2:	4	1	3	7	8	
3:	8	1	2	4	5	7
4:	13	3	6	7		
5:	16	1	3	8		
6:	19	1	4	7	8	
7:	23	2	3	4	6	
8:	27	2	5	6		
	30					

The large matrix is stored as follows:

$s^{11}, s^{22}, s^{33}, s^{44}, s^{55}, s^{66}, s^{77}, s^{88}$

$s_{12}, s_{13}, s_{15}, s_{16}, s_{21}, s_{23}, s_{27}, s_{28}, s_{31}, s_{32}, s_{34}, s_{35}, s_{37}, s_{43}, s_{46},$
$s_{47}, s_{51}, s_{53}, s_{58}, s_{61}, s_{64}, s_{67}, s_{68}, s_{72}, s_{73}, s_{74}, s_{76}, s_{82}, s_{85}, s_{86}$

JMETOD = 11, 12 (combined row/column compact storage scheme)

Length: NHORIZ + NVERT + 3 + $\sum_{i=1}^{NUSOL-NBOUND}$ number of free degrees of freedom connected with i.

where NHORIZ, NVERT depend on the structure of the matrix.

This storage scheme is designed for vector processors, to get a small number of long loops instead of a large number of short loops in the matrix-vector operations. It has no positive effect on scalar processors.

All non-zero off-diagonal elements are compressed to the left, then the rows are sorted to achieve the decreasing length (sorting time grows linearly only).

Then the rows are split into two parts, "long" rows 1..NHORIZ and "short" rows NHORIZ+1.., one of the groups (e.g. "long") may be empty.

The "long" rows are stored in the row-by-row order, the "short" rows are stored in the column-by-column order. Zeros are not stored, so the length of the stored rows/columns varies.

The simple criterion for the choice of the value NHORIZ is to minimize the total number of stored rows and columns, which means to minimize NHORIZ + length-of-the-row(NHORIZ+1).

Such criterion doesn't respect the fact, that the column-by-column storage is less effective because memory conflicts may appear when in the same column there are elements of the same original column of the large matrix. There is no way to estimate the effect for all hardware, at present we penalize the column storage by a heuristic factor 2, minimizing NHORIZ + 2 ∗ length-of-the-row(NHORIZ+1).

Example 24.6.4

See example 24.6.2.

old row	length	new row	new row	length	compressed matrix (new column)					NHORIZ	criterion k=1	criterion k=2
										0	0+5=5	0+2∗5=10
1	4	2	1	5	5	3	2	6	7	1	1+4=5	1+2∗4=9
2	4	3	2	4	1	4	3	7		2	2+4=6	2+2∗4=10
3	5	1	3	4	2	1	8	5		3	3+4=7	3+2∗4=11
4	3	6	4	4	6	2	5	8		4	4+4=8	4+2∗4=12
5	3	7	5	4	3	1	6	4		5	5+3=8	5+2∗3=11
6	4	4	6	3	1	4	5			6	6+3=9	6+2∗3=12
7	4	5	7	3	1	2	8			7	7+3=10	7+2∗3=13
8	3	8	8	3	4	7	3			8	8+0=8	8+2∗0=8

For k=1 the minimum is achieved at NHORIZ=0 or 1, for k=2 the minimum is achieved at NHORIZ=8, second best is NHORIZ=1.

Let NHORIZ = 1 (for the demonstration), then NVERT = length-of-new-row(NHORIZ+1) = 4. The stored row has length 5, the stored column lengths are 7 7 7 4.

In the one-dimensional storage there are NHORIZ, NVERT, the accumulated lengths, and then the new column numbers:

Part 1a: 1 4 0 5 12 19 26 30

Part 1b: 5 3 2 6 7 1 2 6 3 1 1 4 4 1 2 1 4 2 7 3 8 5 6 5 8 3 7 5 8 4

The reverse permutation (from the new to the old numbering) is stored as well:

IOLD: 3 1 2 6 7 4 5 8

The large matrix (after renumbering) is stored in the same order as Part 1b:

$s_{15}\ s_{13}\ s_{12}\ s_{16}\ s_{17}\ \ s_{21}\ s_{32}\ s_{46}\ s_{53}\ s_{61}\ s_{71}\ s_{84}\ \ s_{24}\ s_{31}\ s_{42}\ s_{51}\ s_{64}\ s_{72}\ s_{87}$

$s_{23}\ s_{38}\ s_{45}\ s_{56}\ s_{65}\ s_{78}\ s_{83}\ \ s_{27}\ s_{35}\ s_{48}\ s_{54}$

Notice. The renumbering is an internal matter of this storage scheme. The solver CONGRD (8.5) knows when the permutation should be applied, the matrix-vector routines know how to deal with this storage scheme. This renumbering is hidden for all other software. CONGRD (8.5) takes care of the back permutation of the solution vector also.

Notice. Both the upper and lower triangular parts are stored. One reason is that the permutation doesn't preserve the property to be in the lower part. The second – more serious – reason: the horizontal traversing of the matrix stored in this way is inefficient (large number of short loops) and it eliminates the only advantage of this scheme.

Notice. For matrix-vector multiplication $y_i = s_{ij}x_j$ the first of the stored columns needs to gather the items $s_{21}\ s_{32}\ s_{46}\ s_{53}\ s_{61}\ s_{71}\ s_{84}$ and $x_1\ x_2\ x_6\ x_3\ x_1\ x_1\ x_4$ contributing to $y_{2..8}$ which can cause delays on some hardware, because x_1 appears here three times. This is why the penalty factor k was introduced.

JMETOD = 13 to 16 Simple storage scheme.
The matrix is written as

$$\mathbf{S} = \begin{bmatrix} \mathbf{Q} & \mathbf{G} \\ \mathbf{D} & \mathbf{0} \end{bmatrix}, \tag{24.6.6}$$

with **Q** the velocity matrix, **G** the gradient matrix working on the pressure **p**, but with n_u rows, and **D** the divergence matrix working on the velocity **u**, but with n_p rows.
The first 8 positions of array INTMAT_1a are used for specific storage:

1. istartrowq, starting address of storage for velocity matrix **Q**.
2. istartrowg, starting address of storage for gradient matrix **G**. The storage scheme for the divergence matrix is the same, although it is not necessary that $\mathbf{D} = \mathbf{G}^T$.
3. istartrowdg, starting address of storage for the simple pressure matrix which is formed by $\mathbf{D} \times \mathbf{G}$.
4. n_q, i.e. the size of the matrix Q.
5. n_d, i.e. the size of the matrix D.
6. n_{dg}, i.e. the size of the matrix DG.
7. n_u, i.e. number of velocity unknowns.
8. n_p, i.e. number of pressure unknowns.

The part from istartrowq contains the storage of the matrix **Q**. This storage is a compact storage in the same way as for JMETOD = 5 or 6. Hence the first $n_u + 1$ positions contain the number of velocity degrees of freedom with unknown number smaller than i accumulated. This part is followed by the corresponding column numbers.

The part from istartrowg contains the storage of the matrix **G** or **D**. Again this is a compact storage, where the first $n_p + 1$ positions contain the number of velocity unknowns coupled to the pressure unknown i accumulated. This part is followed by the corresponding column numbers, which refer to the velocity unknowns.

The part from istartrowdg contains the storage of the matrix **DG** contains the same information as for the matrix **Q**, but now for the product matrix $\mathbf{D} \times \mathbf{G}$.

The storage of the large matrix is as follows:
First matrix Q in the standard compact storage format 5 or 6, depending on the symmetry of the velocity matrix.
Next the matrix D in compact storage. In this case, there is no diagonal to be stored.
If the matrix D is not the transpose of G followed by G.
Then the matrix DG is the standard compact storage format. This matrix is symmetrical if D is not the transpose of G, otherwise it is not.
Finally the diagonal of the velocity mass matrix is stored. This is only used for the msimple method.

JMETOD = 20, 21, 22 or 23 The matrix is stored as a full matrix.

JMETOD = 20: the matrix is symmetric
21: the matrix is non-symmetric
22: the matrix is complex symmetric
23: the matrix is general complex

JMETOD = 24 to 27 AL storage scheme.
This is scheme is almost the same as the simple scheme (13,..,16). The only difference is that the matrix **DG** is not present, so no space is reserved for that part.
Instead of the diagonal of the velocity mass matrix, the diagonal of the pressure mass matrix is stored.

Part 2 is only used when NBOUND > 0 or NDEPEN > 0.

First position Part 2 is stored in array IBUFFR. Its starting address can be found by the memory management subroutines using INTMAT(3) as sequence number.

Contents Part 2 contains the connections between prescribed and non-prescribed degrees of freedom and also between degrees of freedom corresponding to periodical boundary conditions.
If IBCMAT=1 the mutual connections between prescribed degrees of freedom is also stored. IBCMAT=1 indicates that also the mutual connections between prescribed degrees of freedom must be stored.

Length: 1 + NBOUND + NDEPEN +IBCMAT $\times$ (1 + NBOUND) $\sum_{i=1}^{NBOUND+NDEPEN}$ number of internal degrees of freedom, connected with prescribed degree of freedom i

Part 2 is divided into two parts, each of which may be considered as 3 separate parts:

Part 2^a contains pointers to the starting position of each prescribed degree of freedom in part 2^b.
Length: NBOUND + 1 + NDEPEN +IBCMAT $\times$ (1 + NBOUND) positions
The first part (2a-1) refers to prescribed degrees of freedom connected to free degrees of freedom (Length: NBOUND+1).
The second part (2a-2) refers to periodical degrees of freedom connected to free degrees of freedom (Length: NDEPEN).
The third part (only used when IBCMAT=1) refers to prescribed degrees of freedom connected to prescribed degrees of freedom (Length: NBOUND+1).
The i^{th} unknown position corresponds to the unknown with new degree of freedom NUSOL - NBOUND + KPROB part h (i). The last position of part 2^a gives the length of part 2^b.

Part 2^b contains for every prescribed unknown the free degrees of freedom it is connected with sequentially.

If IBCMAT = 1, part 2 is extended with information about the matrix S_{pp}. Extra information about S_{pf} is not necessary, since this matrix has the same structure as S_{fp}^T.
In order to store the information about S_{pp} the length of Part 2^a is extended to 2 $\times$ (NBOUND+1) positions. The first NBOUND+1 positions refer to S_{fp} and S_{pf}. The second NBOUND+1 positions to S_{pp}

Part 2^b is extended with a part containing the structure of the matrix S_{pp}. The storage has the same structure as for the matrix S_{fp}.

24.7 Array MATR

Contains integer information concerning the large matrix.
MATR consists of 5 positions:

Pos. 1 JMETOD + 1000 × (IPROB-1), where JMETOD is equal to the parameter indicating the storage scheme of the matrix (JMETOD in subroutine COMMAT (4.5.2)) and IPROB denotes the problem number INTMAT corresponds to.

Pos. 2 103

Pos. 3 MATR1 Memory management sequence number of the first part of the actual information.

Pos. 4 MATR2 |MATR2| gives the memory management sequence number of the decomposition of the large matrix.
When MATR2 < 0, the matrix is positive definite.

Pos. 5 MATR3 Memory management sequence number of the part of the large matrix connected to the boundary conditions.

Actual information

The actual information is divided into three parts: part 1 corresponds to the non-prescribed degrees of freedom, part 2 contains the decomposition of part 1. Usually this decomposition is written on the positions of part 1. Part 3 corresponds to the part of the matrix that is connected with boundary conditions.

Part 1 is stored in array IBUFFR. Its starting address can be found by the memory management subroutines using MATR(3) as sequence number.

Contents The contents of part 1 depend on the parameter JMETOD. See INTMAT. Only the part with respect to free degrees of freedom is stored in this part.

Part 2 is stored in array IBUFFR. Its starting address can be found by the memory management subroutines using MATR(4) as sequence number, provided MATR(4) > 0.

Contents Part 2 contains the decomposition of the large matrix in the case of a direct method. In case of an iterative method, this part may contain the preconditioning matrix.

Part 3 is stored in array IBUFFR. Its starting address can be found by the memory management subroutines using MATR(5) as sequence number, provided MATR(5) > 0.

Contents The matrix elements that connect prescribed with non-prescribed degrees of freedom are stored in part 3. A compact storage is used. Integer information is stored in INTMAT part 2.

If the parameter IBCMAT = 1, part 3 is extended with the matrix parts S_{pf} and S_{pp} in that sequence. Their storage is in accordance to INTMAT part 2.

24.8 Array IVECTR

Contains integer information concerning arrays of special structure.
The following four possibilities are distinguished:

(i) IVECTR corresponds to a vector of special structure, i.e. all degrees of freedom correspond to nodal points. (Type 115)

(ii) IVECTR corresponds to an array that is defined per element, i.e. all degrees of freedom correspond to elements. (Type 116)

(iii) IVECTR corresponds to an array that is defined per node per element, i.e. all degrees of freedom correspond to nodes per element. (Type 126)

(iv) IVECTR corresponds to an array that is defined per integration point per element, i.e. all degrees of freedom correspond to integration points per element. (Type 129)

(v) IVECTR is a special array of type 119, containing functions on curves, which are stored in the sequence x-values, y-values for a number of subcurves. IVECTR is used for example by the subroutines INTP2D and PLOTFN.

(vi) IVECTR is a special array of type 118, containing information about particle trajectories as described in Section 16.8.

IVECTR consists of 5 positions, where the contents depend on the type of vector.

(i) All degrees of freedom correspond to nodes.

Pos. 1 Memory management sequence number of the vector of special structure.

Pos. 2 115 to indicate that this array is a vector of special structure, corresponding to nodal points.

Pos. 3 0 for a real and 1 for a complex vector.

Pos. 4 IVEC + 1000 × (IPROB-1) with:

IVEC denotes the type of vector of special structure ($1 \leq$ IVEC $\leq$ NDTVEC).

IPROB denotes the problem number IVECTR corresponds to.

Pos. 5 NUMDGF, i.e. the number of degrees of freedom in the vector of special structure.

(ii) All degrees of freedom correspond to elements.

Pos. 1 Memory management sequence number of the vector of special structure.

Pos. 2 116 to indicate that this array is a vector of special structure, corresponding to elements.

Pos. 3 0 for a real and 1 for a complex vector.

Pos. 4 NELVEC + 1000 × (IPROB-1) with:

NELVEC denotes the number of degrees of freedom in each element.

IPROB denotes the problem number IVECTR corresponds to.

Pos. 5 NUMDGF, i.e. the number of degrees of freedom in the vector of special structure.

(iii) All degrees of freedom correspond to nodes per element.

Pos. 1 Memory management sequence number of the vector of special structure.

Pos. 2 126 to indicate that this array is a vector of special structure, corresponding to nodes per element.

Pos. 3 0 for a real and 1 for a complex vector.

Pos. 4 NUNKP_NODE + 1000 × (IPROB-1) with:

NUNKP_NODE denotes the number of degrees of freedom per node in each element.

IPROB denotes the problem number IVECTR corresponds to.

Pos. 5 NUMDGF, i.e. the number of degrees of freedom in the vector of special structure.

(iv) All degrees of freedom correspond to integration points per element.

Pos. 1 Memory management sequence number of the vector of special structure.

Pos. 2 129 to indicate that this array is a vector of special structure, corresponding to integration points per element.

Pos. 3 0 for a real and 1 for a complex vector.

Pos. 4 NUNKP_POINT + 1000 × (IPROB-1) with:

NUNKP_POINT denotes the number of degrees of freedom per integration point in each element.

IPROB denotes the problem number IVECTR corresponds to.

Pos. 5 NUMDGF, i.e. the number of degrees of freedom in the vector of special structure.

(v) Special array to store information on curves.

Pos. 1 Memory management sequence number of the vector of special structure.

Pos. 2 119 to indicate that this array contains information of subcurves stored in a special way.

Pos. 3 0 for a real and 1 for a complex vector.

Pos. 4 N: number of subcurves the information consists of.

Pos. 5 Not yet used.

(vi) Special array to store information concerning one particle trace.

Pos. 1 Memory management sequence number of the vector of special structure.

Pos. 2 118 to indicate that this array contains information of a particle trace.

Pos. 3 0 since the array is real.

Pos. 4 NDIM: Dimension of space.

Pos. 5 Length of the double precision array corresponding to IVECTR. This length is equal to (NDIM+1) × npoint_track, with npoint_track the number of points in the trajectory.

For the vectors of type 115 and 116 the array of special structure VECTOR is defined as follows:

Length: NUMDGF or NUMDEL (pos. 5 of IVECTR) real position.

Contents

The vector of special structure is stored sequentially, hence first all degrees of freedom of nodal point number 1 (i) or of element number 1 (ii), then of nodal point number 2 (i) or of element number 2 (ii) etc.

For the vectors of type 126 the array of special structure VECTOR is defined as follows:

Length: NUMDGF (pos. 5 of IVECTR) real positions.

NUMDGF is equal to NUNKP_NODE × NODE_ELEMENTS positions, where NODE_ELEMENTS is $\sum_{i=1}^{N} ELEM$ number of nodes per element.

Contents

The vector of special structure is stored sequentially, hence first all degrees of freedom of relative nodal point number 1 in element 1, then of relative nodal point number 2 in element 1, and so on for all nodes. After element 2 is stored.
Exactly the same sequence as in KMESH part c is used.

The storage of vectors of type 129 is exactly the same as for vectors of type 126, where nodes are replaced by integration points. These vectors may be created by the option DERIVATIVES and reused to define coefficients in the building of matrices. In the case of vectors of type 119 the information is stored in array IBUFFR in the following way:

Contents

The actual information is stored in the following way:
The first N integer positions contain the number of points m_i in each subcurve, followed by information of the various subcurves.
Per subcurve:

m_i x-values
m_i y-values (real or complex)

On a computer using double precision arithmetic, i.e. all 16- and 32-bits machines, where the parameter INTLEN has the value 2, it is necessary that reals in the array IBUFFR should always start on an odd position. Therefore in that case, if N is odd, there is an empty position between the first n integer values and the next reals. This is not the case for the 64-bits computers like CRAY.
So the actual information in IBUFFR is stored as follows:

N positions integers containing the numbers m_i
An empty position if INTLEN = 2 and N is odd.
m_1 x-values corresponding to the first subcurve.
m_1 y-values corresponding to the first subcurve.
m_2 x-values corresponding to the second subcurve.
m_2 y-values corresponding to the second subcurve.

.
.
.

m_N x-values corresponding to the last subcurve.
m_N y-values corresponding to the last subcurve.

In the case of vectors of type 118 the information is stored in array BUFFER. This array may be considered as a two-dimensional double precision array OUTPUT of size (NDIM+1) $\times$ npoint_track. OUTPUT(1,k) contains the time t corresponding to the k^{th} point along the trajectory and OUTPUT(2-NDIM+1,k) contains the corresponding co-ordinates.

24.9 Array MAP

Contains integer information concerning the mapping of one mesh to another one. Array MAP consists of 5 positions:

Pos. 1 mmimap memory management reference to array imap containing the actual integer information about the mapping.

Pos. 2 130

Pos. 3 mmrmap memory management reference to array rmap containing the actual real information about the mapping.

Pos. 4 mmcoor1 memory management reference to array coor1, i.e. the co-ordinates array corresponding to kmesh1. This position is only used for checking.

Pos. 5 mmcoor2 memory management reference to array coor2, i.e. the co-ordinates array corresponding to kmesh2. This position is only used for checking.

The actual arrays IMAP and MAP are defined as follows:

IMAP Integer array of length $(n+2) \times$ NPOINT2, where NPOINT2 is the number of nodal points in KMESH2 (corresponds to the mesh to which the solution must be interpolated). n defines the maximum number of nodes needed for the interpolation per point. At this moment in R^2 $n = 3$, since only linear interpolation in triangles is used.
IMAP may be considered as a two-dimensional array IMAP(NPOINT2,$n + 2$) with the following contents:

imap(i,1) $= 0$ means node i has not been interpolated yet.

imap(i,1) $= m$ (> 0) means node i must be interpolated with m points.

imap(i,2 ... m+1) contain the nodes in the old mesh required for the interpolation.

imap(i,m+2) contains the element sequence number *ielem* in the old mesh from which the points for the interpolation are extracted.
If the new node is outside the original mesh $ielem = 0$.

RMAP Real array of length $n \times$ NPOINT2, with NPOINT2 and n the same as for IMAP.
RMAP may be considered as a two-dimensional array RMAP(NPOINT2,n). RMAP(i,j) (j = 1 (1) m) contains the interpolation coefficients for the mapping in node i.

Hence given the arrays IMAP and RMAP, the mapping of an array U1 to U2, may be defined as follows:

```
do i = 1, NPOINT2
   m = imap(i,1)
   u2(i) = 0
   do j = 1, m
      u2(i) = u2(i) + rmap(i,j) * u1(imap(i,j+1))
   end do
end do
```

Mark that in this example it has been assumed that the number of unknowns in all points of the arrays u1 and u2 is equal to 1 and that the arrays are not renumbered. Otherwise the mapping formula, may be more complicated.

25 Examples

25.1 Introduction

In this chapter we shall give a number of simple examples of programs that are made of subroutines treated in this manual. All these programs are meant to replace the standard program SEPCOMP. Although in a number of cases SEPCOMP itself is capable of handling these examples, this chapter is meant as an introduction to more complicated problems.

The following examples are available in this chapter:

25.2 Examples of linear stationary problems

25.2.1 A potential problem in a L-shaped region

The example treated in this section is completely identical to the example treated in Section 7.1 of the Introduction manual. All input files are exactly the same as in the Introduction, only the main program SEPCOMP is replaced by a user written main program, just to show the structure of SEPCOMP.
All files are available in the SEPRAN example directory and can be transported to your own directories by the command:

```
sepgetex potential1
```

The structure of the main program is as follows:

```
      program potential1
      implicit none
      integer lenkmesh, lenkprob
      parameter ( lenkmesh=500, lenkprob = 500 )
      integer kmesh(lenkmesh), kprob(lenkprob), intmat(5), iinstr(2),
     +        isol(5), iinprs(1), iinsol(1), iinout(1), iuser(1),
     +        istop
      double precision time, user(1)

      data kmesh /lenkmesh*0/, kprob /lenkprob*0/, isol/5*0/,
     +     intmat/5*0/

!     --- Initialize variable length arrays

      kmesh(1) = lenkmesh
      kprob(1) = lenkprob

!     --- Start SEPRAN and read mesh

      iinstr(1) = 2
      iinstr(2) = 1
      call sepstn ( kmesh, kprob, intmat, iinstr )

!     --- Fill essential boundary conditions in ISOL

      iinprs(1) = 0
      call presbc ( kmesh, kprob, isol, iinprs )

!     --- Build and solve linear system

      iinsol(1) = 0
      call linprb ( kmesh, kprob, intmat, isol, iinsol )

!     --- Write results to file sepcomp.out

      iinout(1) = 0
      time = 0d0
      call outsol ( kmesh, kprob, isol, iinout, time, iuser, user )

!     --- Stop SEPRAN
```

```
istop = 0
call finish ( istop )

end
```

25.2.2 A potential problem in a L-shaped region with refinement

The example treated in this section is the same as in Section 25.2.1. The only difference is that in this example the mesh is refined two times, each producing a grid twice as fine as the preceding one. The maximal difference of the potential between two succeeding meshes is computed and printed. At the end of the run the then computed mesh is written to the file `meshoutput`. Since subroutine SEPSTN opens and closes this file, this implies that the previous mesh is overwritten. In this way the postprocessing may be performed on the finest mesh.

The input files for this program are identical to the input files treated in Section 7.1 of the SEPRAN INTRODUCTION. A useful extra option might be to plot the mesh in the postprocessing file.
All files are available in the SEPRAN example directory and can be transported to your own directories by the command:

```
sepgetex potaccur
```

The main program in this case has the following shape:

```
      integer iinstr(2), iinfil(1), iinbld(1), inpsol(1), iinout(1),
     +        nref, iref, iread, ichois, iinput(1), iinprs(1)

      include 'SPcommon/cmcdpi'
      include 'SPcommon/cmachn'

      data kmesh/lenkmesh*0/, kmeshcp/lenkmesh*0/, kprob/lenkprob*0/,
     +     kprobcp/lenkprob*0/, isol/5*0/, isolcp/5*0/, isolsav/5*0/,
     +     intmat/5*0/, iuser/leniuser*0/, user/lenuser*0d0/, matr/5*0/,
     +     irhsd/5*0/

!     --- Fill first positions of variable length arrays

      kmesh(1) = lenkmesh
      kmeshcp(1) = lenkmesh
      kprob(1) = lenkprob
      kprobcp(1) = lenkprob
      iuser(1) = leniuser
      user(1) = lenuser

!     --- start sepran, read mesh and input

      iinstr(1) = 2
      iinstr(2) = 1
      call sepstn ( kmesh, kprob, intmat, iinstr )

!     --- Loop over refinements

      nref = 3
      do iref = 1, nref

!     --- fill essential boundary conditions

         iinprs(1) = 0
         call presbc ( kmesh, kprob, isol, iinprs )

!        --- fill coefficients
```

```
         iinfil(1) = 0
         call fillcf ( iuser, user, kprob, kmesh, iinfil )

!        --- build matrix and right-hand side

         iinbld(1) = 0
         call build ( iinbld, matr, intmat, kmesh, kprob, irhsd,
     +                massmt, isol, isol, iuser, user )

!        --- solve linear system of equations

         inpsol(1) = 0
         iread = 1
         call solvel ( inpsol, rinsol, matr, isol, irhsd, intmat,
     +                 kmesh, kprob, iread )
         if ( iref>1 ) then

!        --- iref > 1, compute difference

            call intmsh ( 1, kmesh, kmeshcp, kprob, kprobcp, isol,
     +                    isolcp )
            call diffvc ( 1, isolsav, isolcp, kprobcp, difmax )
            write(irefwr,100) iref, difmax
100         format ( ' maximum difference in refinement ', i1,' is',
     +               d13.3 )

         end if

         if ( iref<nref ) then

!        --- Copy and refine mesh

            call mshcopy ( kmesh, kmeshcp )
            call procopy ( kprob, kprobcp )
            call copyvc ( isol, isolsav )
            ichois = 1
            call refine ( kmesh, ichois, iinput )
            call probdf ( 3, kprob, kmesh, iinput )
            call commat ( 1, kmesh, kprob, intmat )

         end if

      end do

!     --- write solution to files for postprocessing

      iinout(1) = 0
      call outsol ( kmesh, kprob, isol, iinout, 0d0, iuser,
     +              user )

      if ( nref>1 ) then

!     --- overwrite meshoutput with new mesh

         if ( iref10<0 ) then
```

```
            open ( unit=-iref10, file='meshoutput', form='unformatted' )
         else
            open ( unit=iref10, file='meshoutput', form='formatted' )
         end if  ! ( iref10<0 )
         call meshwr ( iref10, kmesh )

      end if

!     --- Close all files

      call finish(0)

      end
```

25.3 Examples of non-linear stationary problems

25.3.1 Stationary isothermal laminar Newtonian flow in a T-shaped region

The example treated in this section is completely identical to the example treated in Section 7.3 of the Introduction manual. All input files are exactly the same, only the main program SEPCOMP is replaced by a user written main program, just to show the structure of SEPCOMP.

```
      program navstokes
      implicit none
      integer lenkmesh, lenkprob
      parameter ( lenkmesh=500, lenkprob = 500 )
      integer kmesh(lenkmesh), kprob(lenkprob), intmat(5), iinstr(2),
     +        isol(5), iinprs(1), iinsol(1), iinout(1), iuser(1),
     +        istop
      double precision time, user(1)

      data kmesh /lenkmesh*0/, kprob /lenkprob*0/, isol/5*0/,
     +     intmat/5*0/

c     --- Initialize variable length arrays

      kmesh(1) = lenkmesh
      kprob(1) = lenkprob

c     --- Start SEPRAN and read mesh

      iinstr(1) = 2
      iinstr(2) = 1
      call sepstn ( kmesh, kprob, intmat, iinstr )

c     --- Fill essential boundary conditions in ISOL

      iinprs(1) = 0
      call presbc ( kmesh, kprob, isol, iinprs )

c     --- Build and solve non-linear system

      iinsol(1) = 0
      call nlnprb ( kmesh, kprob, intmat, isol, iinsol )

c     --- Write results to file sepcomp.out

      iinout(1) = 0
      time = 0d0
      call outsol ( kmesh, kprob, isol, iinout, time, iuser, user )

c     --- Stop SEPRAN

      istop = 0
      call finish ( istop )

      end
```

25.4 Examples of time-dependent problems

In this section we shall treat some examples of time dependent problems. The following examples are available

25.4.1 treats the use of subroutine TIMPRB.

25.4.2 deals with subroutine SOLTIM in case of a diagonal mass matrix.

25.4.3 is concerned with subroutine SOLTIM in case of a non-diagonal mass matrix.

25.4.4 gives an example of subroutine SOLTIM in case of a a second order equation, i.e. containing second order time-derivatives.

25.4.1 An example of the use of subnroutine TIMPRB

This section is not available yet.

25.4.2 An example of the use of subroutine SOLTIM for a diagonal mass matrix

As an example of the use of SOLTIM we consider exactly the same example as given in Section 6.4.1 of the Users Manual.
To create the mesh program SEPMESH is used with input file examtim1.msh:

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

In the main program subroutine SOLTIM is called and hence subroutine FILTIM must be provided. The program itself is straight forward. SOLTIM is used with Crank-Nicolson with self-selecting step size (ichtim=1). Output is produced for time $t = 0.1, 0.2, \ldots, 1.0$.
The main program examtim1 has the following shape:

```
      program examtim1
      implicit none

      double precision t, tout, tstep, tend, t0, rtime
      integer iflag, ktime
      common /ctimen/ t, tout, tstep, tend, t0, rtime(5), iflag,
     +                ktime(9)

      integer kmesh(100), kprob(200), intmat(5), isol(5), iuser(100),
     +        ichtim, icount, itytim, iincrt(1), iinout(1), iinstr(2)
      double precision user(100), epsusr

!     --- Standard SEPRAN initialisations

      kmesh(1) = 100
      kprob(1) = 200
      user(1)  = 100
      iuser(1) = 100
      kmesh(3) = 0
      kprob(3) = 0
      user(3)  = 0
      iuser(3) = 0

!     --- Start SEPRAN  (read mesh etc.)
```

```
      iinstr(1) = 2
      iinstr(2) = 1
      call sepstn ( kmesh, kprob, intmat, iinstr )

!     --- fill initial temperature in isol

      t      = 0d0
      iincrt(1) = 0
      call creatn ( iincrt, kmesh, kprob, isol )

!     --- Write initial condition to file sepcomp.out

      iinout(1) = 0
      call outsol ( kmesh, kprob, isol, iinout, t, iuser, user )

!     --- set parameters for subroutine soltim:

      itytim = 1
      ichtim = 1
      epsusr = 1d-3

!     --- compute solution for t = 0.1, 0.2, . . . , 1.0

      tend = 1d0
      do icount = 1, 10
         tout = icount*0.1d0
         call soltim ( itytim, ichtim, kmesh, kprob, intmat, isol,
     +                 epsusr, user, iuser)

!      --- write solution to file sepcomp.out

         call outsol ( kmesh, kprob, isol, iinout, t, iuser, user )

      end do

!     --- Close SEPRAN

      call finish(0)
      end


!     --- function func for the initial condition


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

      func = sin(x)*sin(y)

      end


!     --- function funcbc for essential boundary conditions
```

```
      function funcbc ( ichoice, x, y, z )
      implicit none
      double precision funcbc, x, y, z
      integer ichoice
      double precision t, tout, tstep, tend, t0, rtimdu
      integer iflag, icons, itimdu
      common /ctimen/ t, tout, tstep, tend, t0, rtimdu(5), iflag,
     +                icons, itimdu(8)

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

      end


!     --- subroutine filtim for the control of the time-dependent process


      subroutine filtim ( kmesh, kprob, intmat, isol, matrs, matrm,
     +                    irhsd, islold, user, iuser )
      implicit none
      double precision user(*)
      integer kmesh(*), kprob(*), intmat(*), isol(*), islold(*),
     +        matrs(*), matrm(*), irhsd(*), iuser(*)
      integer ictime, iinbld(4), iinprs(1), iinfil(1)
      save ictime
      data ictime /0/

!     --- User written subroutine for the incorporation of boundary
!         conditions and the determination of the matrices M and S
!         Example : Heat equation on the unit square.


!     --- essential boundary conditions

      iinprs(1) = 0
      call presbc ( kmesh, kprob, isol, iinprs )

!     --- Fill matrices M and S and right-hand side f
!         Since the matrices are time-independent this is only
!         necessary at t = 0

      if ( ictime==0 ) then

!     --- Information is only read the first step
!         The parameter ictime is used to distinguish between first and
!         next step

         iinfil(1) = 0
         call fillcf ( iuser, user, kprob, kmesh, iinfil )

!        --- Do not take boundary conditions into account
!            Mass matrix is diagonal
```

```
         iinbld(1) = 4
         iinbld(2) = 1
         iinbld(3) = 1
         iinbld(4) = 1
         call build ( iinbld, matrs, intmat, kmesh, kprob,
     +                irhsd, matrm, isol, islold, iuser, user )
         ictime = 1

       end if
       end
```

The corresponding input file examtim1.prb is for example:

```
*  examtim1.prb
constants
   vector_names
      potential
end

*  problem definition for time-dependent heat equation
*  linear triangles  type number 800
problem
   types
      elgrp1 = 800
   essbouncond
      curves (c1,c4)
end
*
*  Definition of matrix structure
*
matrix
   symmetric
end
*
*  Define initial conditions
*
create vector
   func = 1
end
*
*  Input for subroutine outsol
*
output
end
*
*  Essential boundary conditions
*  Only at t=0 the information is read
*
essential boundary conditions
   curves (c1,c4),(func=1)
end
*
*  Definition of coefficients for the heat equation (t=0 only)
*
coefficients
```

```
   elgrp1(nparm=20)
      coef6  = 0.5               # a11 = 0.5
      coef9  = coef 6            # a22 = 0.5
      coef17 = 1                 # rho = 1
end
end_of_sepran_input
```

Finally post processing may be performed by program seppost with input file examtim1.pst:

```
* examtim1.pst
*
* input for seppost
*
postprocessing
   time = (0,1)
      print potential
      plot contour potential, minlevel = 0, maxlevel = 1
   time history plot point(.5,.5) potential
end
```

25.4.3 An example of the use of subroutine SOLTIM for a non-diagonal mass matrix

This example is example is completely identical to the example in Section 25.4.2. The only difference is that the diagonal mass matrix is replaced by a non-diagonal mass matrix. This has only a minor influence in the call of BUILD (7.2) in subroutine FILTIM.
All input files remain the same as in Section 25.4.2, only the source of the main program examtim2 differs a little bit from 25.4.2:

```
      program examtim2
      implicit none

      double precision t, tout, tstep, tend, t0, rtime
      integer iflag, ktime
      common /ctimen/ t, tout, tstep, tend, t0, rtime(5), iflag,
     +                ktime(9)

      integer kmesh(100), kprob(200), intmat(5), isol(5), iuser(100),
     +        ichtim, icount, itytim, iincrt(1), iinout(1), iinstr(2)
      double precision user(100), epsusr

!     --- Standard SEPRAN initialisations

      kmesh(1) = 100
      kprob(1) = 200
      user(1)  = 100
      iuser(1) = 100
      kmesh(3) = 0
      kprob(3) = 0
      user(3)  = 0
      iuser(3) = 0

!     --- Start SEPRAN  (read mesh etc.)

      iinstr(1) = 2
      iinstr(2) = 1
      call sepstn ( kmesh, kprob, intmat, iinstr )

!     --- fill initial temperature in isol

      t      = 0d0
      iincrt(1) = 0
      call creatn ( iincrt, kmesh, kprob, isol )

!     --- Write initial condition to file sepcomp.out

      iinout(1) = 0
      call outsol ( kmesh, kprob, isol, iinout, t, iuser, user )

!     --- set parameters for subroutine soltim:

      itytim = 1
      ichtim = 1
      epsusr = 1d-3
```

```
!     --- compute solution for t = 0.1, 0.2, . . . , 1.0

      tend = 1d0
      do icount = 1, 10
         tout = icount*0.1d0
         call soltim ( itytim, ichtim, kmesh, kprob, intmat, isol,
     +                 epsusr, user, iuser)

!      --- write solution to file sepcomp.out

         call outsol ( kmesh, kprob, isol, iinout, t, iuser, user )

      end do

!     --- Close SEPRAN

      call finish(0)
      end


!     --- function func for the initial condition


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

      func = sin(x)*sin(y)

      end


!     --- function funcbc for essential boundary conditions


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

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

      end


!     --- subroutine filtim for the control of the time-dependent process


      subroutine filtim ( kmesh, kprob, intmat, isol, matrs, matrm,
     +                    irhsd, islold, user, iuser )
```

```
      implicit none
      double precision user(*)
      integer kmesh(*), kprob(*), intmat(*), isol(*), islold(*),
     +        matrs(*), matrm(*), irhsd(*), iuser(*)
      integer ictime, iinbld(4), iinprs(1), iinfil(1)
      save ictime
      data ictime /0/

!     --- User written subroutine for the incorporation of boundary
!         conditions and the determination of the matrices M and S
!         Example : Heat equation on the unit square.


!     --- essential boundary conditions

      iinprs(1) = 0
      call presbc ( kmesh, kprob, isol, iinprs )

!     --- Fill matrices M and S and right-hand side f
!         Since the matrices are time-independent this is only
!         necessary at t = 0

      if ( ictime==0 ) then

!     --- Information is only read the first step
!         The parameter ictime is used to distinguish between first and
!         next step

         iinfil(1) = 0
         call fillcf ( iuser, user, kprob, kmesh, iinfil )

!        --- Do not take boundary conditions into account
!            Mass matrix is non-diagonal

         iinbld(1) = 4
         iinbld(2) = 1
         iinbld(3) = 1
         iinbld(4) = 2
         call build ( iinbld, matrs, intmat, kmesh, kprob,
     +                irhsd, matrm, isol, islold, iuser, user )
         ictime = 1

      end if
      end
```

25.4.4 An example of the use of subroutine SOLTIM for a second order equation

In this artificial example we consider a second order time-derivative, i.e. the wave equation on the unit square $\Omega = (0,1) \times (0,1)$ with boundary Γ:

$$\frac{\partial^2 u}{\partial t^2} - \Delta u = f \quad x \in \Omega \tag{25.4.4.1}$$

with initial conditions:

$$u(\mathbf{x},0) = x + 3y, \qquad \frac{\partial u}{\partial t}(\mathbf{x},0) = 0 \quad x \in \Omega \tag{25.4.4.2}$$

and boundary conditions:

$$u(\mathbf{x},t) = cos(t)(x+3y), \quad x \in \Gamma \tag{25.4.4.3}$$

In this special case the exact solution is known and is equal to

$$u(\mathbf{x},t) = cos(t)(x+3y) \tag{25.4.4.4}$$

The mere purpose of this example is just to show, how to use subroutine FILTIM for second order problems.
The mesh used in this example is exactly the same as in example 25.4.2, so we do not recall the mesh input file.
The main problem examtim3 has the following shape:

```
      program examtim3
      implicit none

      double precision t, tout, tstep, tend, t0, rtime
      integer iflag, ktime
      common /ctimen/ t, tout, tstep, tend, t0, rtime(5), iflag,
     +                ktime(9)
      integer irefwr, irefre, irefer
      common /cmcdpi/ irefwr, irefre, irefer

      integer kmesh(100), kprob(200), intmat(5), isol(5,2), iuser(100),
     +        ichtim, icount, itytim, iincrt(3), iinout(1), iinstr(2),
     +        iinvec(6), iex(5)
      double precision user(100), rinvec(1), epsusr

!     --- Standard SEPRAN initialisations

      kmesh(1) = 100
      kprob(1) = 200
      user(1)  = 100
      iuser(1) = 100
      kmesh(3) = 0
      kprob(3) = 0
      user(3)  = 0
      iuser(3) = 0

!     --- Start SEPRAN  (read mesh etc.)

      iinstr(1) = 2
      iinstr(2) = 1
      call sepstn ( kmesh, kprob, intmat, iinstr )
```

```
!     --- fill initial field in isol

      t      = 0d0
      iincrt(1) = 0
      call creatn ( iincrt, kmesh, kprob, isol )

!     --- Write initial condition to file sepcomp.out

      iinout(1) = 0
      call outsol ( kmesh, kprob, isol, iinout, t, iuser, user )

!     --- set parameters for subroutine soltim:

      itytim = 1
      ichtim = 11
      tstep = 0.05d0
      epsusr = 0d0

!     --- compute solution for t = 0.1, 0.2, . . . , 1.0

      tend = 1d0
      do 200 icount = 1, 10
         tout = icount*0.1d0
         call soltim ( itytim, ichtim, kmesh, kprob, intmat, isol,
     +                 epsusr, user, iuser)

!      --- write solution to file sepcomp.out

         call outsol ( kmesh, kprob, isol, iinout, t, iuser, user )

!        --- Compare with exact solution

         iincrt(1) = 3
         iincrt(2) = 0
         iincrt(3) = 2
         call creatn ( iincrt, kmesh, kprob, iex )
         iinvec(1) = 6
         iinvec(2) = 5
         iinvec(3) = 0
         iinvec(4) = 1
         iinvec(5) = 1
         iinvec(6) = 3
         call manvec ( iinvec, rinvec, isol, iex, isol, kmesh, kprob )
         write ( irefwr, 100 ) t, rinvec(1)
100      format( 'Maximal difference at t = ',f5.1,' = ', d12.2 )

200   continue

!     --- Close SEPRAN

      call finish(0)
      end


!     --- function func for the initial condition
```

```
      function func ( ichoice, x, y, z )
      implicit none
      double precision func, x, y, z
      integer ichoice
      double precision t, tout, tstep, tend, t0, rtimdu
      integer iflag, icons, itimdu
      common /ctimen/ t, tout, tstep, tend, t0, rtimdu(5), iflag,
     +                icons, itimdu(8)

      func =  cos(t) * ( x + 3d0 * y )

      end


!     --- function funccf for the right-hand side:

      function funccf ( ichoice, x, y, z )
      implicit none
      double precision funccf, x, y, z
      integer ichoice
      double precision t, tout, tstep, tend, t0, rtimdu
      integer iflag, icons, itimdu
      common /ctimen/ t, tout, tstep, tend, t0, rtimdu(5), iflag,
     +                icons, itimdu(8)

      funccf = -cos(t) * ( x + 3d0 * y )

      end


!     --- function funcbc for essential boundary conditions


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

      funcbc = cos(t+tstep) * ( x + 3d0 * y )

      end


!     --- subroutine filtim for the control of the time-dependent process


      subroutine filtim ( kmesh, kprob, intmat, isol, matrs, matrm,
     +                    irhsd, islold, user, iuser )
      implicit none
      double precision user(*)
```

```
      integer kmesh(*), kprob(*), intmat(*), isol(*), islold(*),
     +        matrs(*), matrm(*), irhsd(*), iuser(*)
      integer ictime, iinbld(4), iinprs(1), iinfil(1)
      save ictime
      data ictime /0/

!     --- User written subroutine for the incorporation of boundary
!         conditions and the determination of the matrices M and S
!         Example : Heat equation on the unit square.


!     --- essential boundary conditions

      iinprs(1) = 0
      call presbc ( kmesh, kprob, isol, iinprs )

!     --- Fill matrices M and S and right-hand side f
!         Since the matrices are time-independent the matrices
!         will only be filled at t = 0

      if ( ictime==0 ) then

!     --- Information is only read the first step
!         The parameter ictime is used to distinguish between first and
!         next step

         iinfil(1) = 0
         call fillcf ( iuser, user, kprob, kmesh, iinfil )

!        --- Do not take boundary conditions into account
!            Do not compyte the right-hand side
!            Mass matrix is diagonal

         iinbld(1) = 4
         iinbld(2) = 14
         iinbld(3) = 1
         iinbld(4) = 1
         call build ( iinbld, matrs, intmat, kmesh, kprob,
     +                irhsd, matrm, isol, islold, iuser, user )
         ictime = 1

      end if

!     --- Build the time-dependent right-hand side

      iinbld(1) = 3
      iinbld(2) = 2
      iinbld(3) = 1
      call build ( iinbld, matrs, intmat, kmesh, kprob,
     +             irhsd, matrm, isol, islold, iuser, user )
      end
```

Mark that in this example isol consists of two arrays actually, one to store the solution and one to store the time derivative at $t = 0$.
The exact solution is known and stored in the array corresponding to iex. The difference between

exact and numeric solution is computed and printed at time levels $t = 0.1, 0.2, \ldots, 1$.
Since the matrices do not depend on time, whereas the right-hand side does, matrices and right-hand side are computed in different steps in FILTIM.

The corresponding input file examtim3.prb reads:

```
*  examtim3.prb
constants
   vector_names
      potential
end

*  problem definition for time-dependent wave equation
*  linear triangles  type number 800
problem
   types
      elgrp1 = 800
   essbouncond
      curves (c1,c4)
end
*
*  Definition of matrix structure
*
matrix
   storage_method = compact, symmetric
end
*
*  Define initial conditions
*
create vector 1, sequence_number = 1
   func = 1
create vector 2
   value = 0
end
*
*  Input for subroutine outsol
*
output
end
*
*  Essential boundary conditions
*  Only at t=0 the information is read
*
essential boundary conditions
   curves (c1,c4),(func=1)
end
*
*  Definition of coefficients for the heat equation (t=0 only)
*
coefficients
   elgrp1(nparm=20)
      coef6  = 1                  # a11 = 1
      coef9  = coef 6             # a22 = 1
      coef16 = func = 1           # f
      coef17 = 1                  # rho = 1
end
```

```
*
*  Definition of exact solution
*
create vector, sequence_number = 2
   func = 1
end
end_of_sepran_input
```

Postprocessing may be performed in exactly the same way as in the examples in the Sections 25.4.2 and 25.4.3.

25.5 Examples of eigenvalue problems

25.5.1 An example of the use of subroutine EIGVAL

As an example of the use of EIGVAL we compute the 3 smallest eigenvalues and corresponding eigenvectors of the Laplace equation on a L-shaped region.
Consider the eigenvalue problem associated to the Laplace operator:

$$-\Delta u \;=\; \lambda u \qquad x \in \Omega \tag{25.5.1.1}$$

with

$$u \;=\; 0 \qquad \text{at the boundary } \Gamma \text{ of } \Omega. \tag{25.5.1.2}$$

Our purpose is to find the 3 smallest eigenvalues of the problem 25.5.1.1, 25.5.1.2 and their corresponding non-trivial eigenfunctions u. Discretization of 25.5.1.1, 25.5.1.2 by the finite element method yields the following generalised eigenvalue problem:

$$\mathbf{S}\,\mathbf{u} \;=\; \lambda\,\mathbf{M}\,\mathbf{u}, \tag{25.5.1.3}$$

with $\mathbf{S}$ and $\mathbf{M}$ positive definite. The region Ω is sketched in Figure 25.5.1.1 The discretization

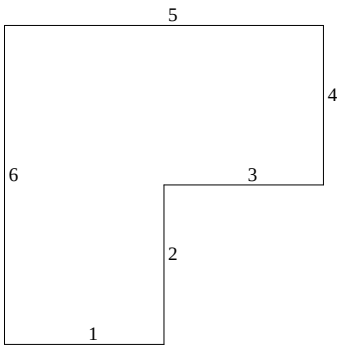


Figure 25.5.1.1: L-shaped region for the eigenvalue problem

is performed with linear triangles and the standard Laplacian equation described in the manual Standard Problems Section 3.1 (type number 800).

To create the mesh, program SEPMESH is used with input file eigval.msh:

```
*
*  eigval.msh
*
*  mesh for 2D eigenvalue example
*  L-shaped region
*
*         P6                C5                          P5
*         *-------------------------------------------*
*         |                                           |
*         |                                           |
*         |                                           | C4
*       C6|                    P3                     |
*         |                     *---------------------*
*         |                     |        C3           P4
*         |                     |
*         |                     |C2
*         |                     |
*         *---------------------*
*         P1          C1        P2
*
mesh2d
   coarse(unit=0.1)
   points
      p1=(0,0,1)
      p2=(1,0,1)
      p3=(1,1,1)
      p4=(2,1,1)
      p5=(2,2,1)
      p6=(0,2,1)
   curves
      c1=cline1(p1,p2)
      c2=cline1(p2,p3)
      c3=cline1(p3,p4)
      c4=cline1(p4,p5)
      c5=cline1(p5,p6)
      c6=cline1(p6,p1)
   surfaces
      s1=general3(c1,c2,c3,c4,c5,c6)
   plot
end
```

The mesh created by sepmesh is shown in Figure 25.5.1.2.

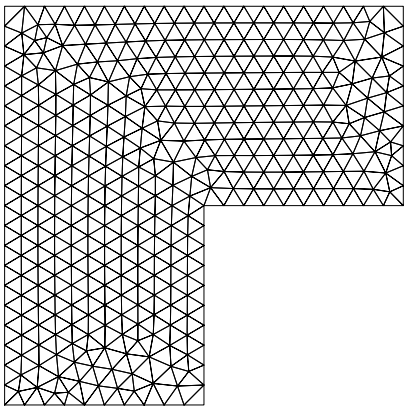


Figure 25.5.1.2: Mesh of L-shaped region generated by GENERAL

In this case it is not yet possible to use program SEPCOMP. However, the following simple main program (eigvalex) may be used to compute the eigenvalues. The source itself is self explaining.

```
      program eigvalex
c ======================================================================
c
c
c        programmer     Guus Segal
c        version 1.0    date   07-01-96
c
c ***********************************************************************
c
c                          DESCRIPTION
c
c     example for the computation of eigenvalues
c
c ***********************************************************************
c
c                        KEYWORDS
c
c     eigenvalue
c     eigenvector
c ***********************************************************************
c
c                        INPUT / OUTPUT    PARAMETERS
c
      implicit none

c   -
c ***********************************************************************
```

```
c
c                      COMMON BLOCKS
c
      integer irefwr, irefre, irefer
      common /cmcdpi/ irefwr, irefre, irefer
      save /cmcdpi/
c
c                 /cmcdpi/
c    Unit numbers to use for certain standard in- and output files.
c
c    irefwr     Unit number to use for "normal" writes
c    irefre     Same for standard reads (mostly keyboard input)
c    irefer     Same for error messages
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
c ****************************************************************
c
c                    LOCAL PARAMETERS
c
      integer kmesh(100), kprob(100), intmat(5), iinbld(10), iinout(1),
     +        istiff(5), mass(5), icheig(10), ivec(5,3), iinstr(2),
     +        iinfil(1), idummy, i, numeig, iuser(100), irhsd(5)
      double precision user(100), eval(10), time

c    eval     Variable length array in which the eigenvalues are stored
c    i        General loop variable
c    icheig   Input array for subroutine EIGVAL
c             ICHEIG is filled as follows:
c             Pos. 1:  1, i.e. eigenvectors must be computed
c                  2:  1, i.e. information about the iteration must be printed
c                  3-8: 0, these positions are not yet used
c    idummy   Dummy parameter
c    iinbld   Input array for subroutine BUILD
c             IINBLD is filled as follows:
c             Pos. 1:  4, i.e. positions 1-4 are filled
c                  2: 14, matrix is build, but not the right-hand side
c                  3:  1, the effect of boundary conditions is not taken into
c                         account
c                  4:  1, the mass matrix is diagonal
c    iinfil   Input array for subroutine FILLCF
c             IINFIL is filled as follows:
c             Pos. 1:  0, i.e. only position 1 is filled
c    iinout   Input array for subroutine OUTSOL
c             IINFIL is filled as follows:
c             Pos. 1:  0, i.e. only position 1 is filled
c    iinstr   Input array for subroutine SEPSTN
c             IINSTR is filled as follows:
c             Pos. 1:  2, i.e. positions 1-2 are filled
c             Pos. 2:  1, i.e. all input until end_of_sepran_input is read
c    intmat   Standard SEPRAN array containing information of the storage
c             scheme of the matrix
c    irhsd    Array containing information about the right-hand side
c    istiff   Contains information about the stiffness matrix
c    iuser    Variable length user array to be filled by FILCOF which
c             contains information about the coefficients
c    ivec     Contains information about the three eigenvectors
```

```
c             ivec(j,i), j=1,5 relates to eigenvector i
c    kmesh    Standard SEPRAN array containing information of the mesh
c    kprob    Standard SEPRAN array containing information of the problem
c    mass     Contains information about the mass matrix
c    numeig   Number of eigenvalues and eigenvectors to be computed
c    time     Dummy parameter
c    user     Variable length double precision user array to be filled by FILCOF
c             which contains information about the coefficients
c ******************************************************************
c
c             SUBROUTINES CALLED
c
c     BUILD   Build matrices
c     EIGVAL  Compute eigenvalues and eigenvectors
c     FILLCF  Fill parameters for equations
c     FINISH  Stop SEPRAN
c     OUTSOL  Write eigenvectors to backing storage
c     SEPSTN  Start SEPRAN and read mesh
c
c ******************************************************************
c
c                    I/O
c
c    This program reads the standard SEPRAN files.
c    It assumes that the mesh has been generated by program SEPMESH
c    with input file eigval.msh and output file meshoutput
c    It reads input from the standard input file (eigval.prb)
c    It produces two files: sepcomp.inf and sepcomp.out for program SEPPOST
c    Program SEPPOST requires the input file eigval.pst
c ******************************************************************
c
c                    ERROR MESSAGES
c
c   -
c ******************************************************************
c
c                    PSEUDO CODE
c
c   Fill length of variable length arrays
c   Start SEPRAN and read mesh
c   Fill coefficients
c   Build matrices
c   Compute eigenvalues and eigenvectors
c   Print eigenvalues
c   Write eigenvectors for postprocessing
c ==================================================================
c
c     --- Fill length of variable length arrays

      kmesh(1) = 100
      kprob(1) = 100
      user(1)  = 100
      iuser(1) = 100
      eval(1)  = 10
      kmesh(3) = 0
```

```
      kprob(3) = 0
      user(3)  = 0
      iuser(3) = 0
      eval(3)  = 0

c     --- Start SEPRAN and read mesh

      iinstr(1) = 2
      iinstr(2) = 1
      call sepstn ( kmesh, kprob, intmat, iinstr )

c     --- fill coefficients for Laplace equation

      iinfil(1) = 0
      call fillcf ( iuser, user, kprob, kmesh, iinfil )

c     --- build mass matrix and stiffness matrix
c         not the right-hand side!

      iinbld(1) = 4
      iinbld(2) = 14
      iinbld(3) = 1
      iinbld(4) = 1
      call build ( iinbld, istiff, intmat, kmesh, kprob, irhsd, mass,
     +             ivec, idummy, iuser, user )

c     --- compute eigenvalues and eigenvectors

      icheig(1) = 1
      icheig(2) = 1
      do 110 i = 3, 8
         icheig(i) = 0
110   continue
      numeig = 3
      call eigval ( eval, numeig, ivec, istiff, intmat, mass, intmat,
     +              kprob, icheig )

c     --- output

      do 200 i = 1, numeig
         write ( irefwr, 120 ) i, eval(i+5)
120      format(' eigenvalue',i4,':',e12.5)
200   continue

c     --- write eigenvectors to "sepcomp.out"

      time = 0d0
      iinout(1) = 0
      call outsol ( kmesh, kprob, ivec, iinout, time, iuser, user )

c     --- Stop SEPRAN

      call finish(0)
      end
```

eigvalex requires a standard input file, for example eigval.prb:

```
*
*  File:  eigval.prb
*
*  problem definition for 2D eigenvalue example
*  L-shaped region
*
*  Problem definition:
*
problem
   types
      elgrp1 = (type=800)
   essbouncond
      curves0(c1,c6)
end

*  The matrix is a compact symmetric matrix

matrix
   method = 5
end

*  Define 20 coefficients for the Laplacian equation

coefficients
   elgrp1(nparm=20)
      icoef1 = 1                # diagonal mass matrix
      coef6  = 1                # a11 = 1 for laplace
      coef9  = 1                # a22 = 1 for laplace
      coef17 = 1                # rho = 1 (mass matrix)
end

*  write the three eigenvectors

output
   write 3 solutions
end
end_of_sepran_input
```

To visualise the results program SEPPOST is used with the following input file (eigval.pst)

```
*
*  File:  eigval.pst
*
*  postprocessing input for 2D eigenvalue example
*  L-shaped region
*
postprocessing

*  Define the names of the vectors

   name v0 = eigenvector 1
   name v1 = eigenvector 2
   name v2 = eigenvector 3
```

```
*  Print the three vectors

   print v0
   print v1
   print v2

*  Define plot identification

   plot identification, text='Example of eigenvalue problem', origin=(3,18)

*  Make contour plots and coloured level plots of all three eigenvectors

   plot contour v0
   plot coloured contour v0
   plot contour v1
   plot coloured contour v1
   plot contour v2
   plot coloured contour v2
end
```

Hence, the following steps are used:

```
   sepmesh eigval.msh
   sepdisplay or sepview
   seplink eigvalex
   eigvalex < eigval.prb > eigval.out
   seppost eigval.pst
   sepdisplay or sepview
```

The Figures 25.5.1.3, 25.5.1.4 and 25.5.1.5 show the isolines of the first three eigenvectors.

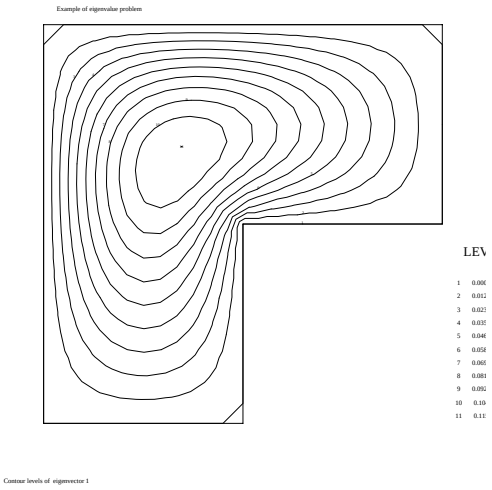


Figure 25.5.1.3: Contour plot of first eigenvector

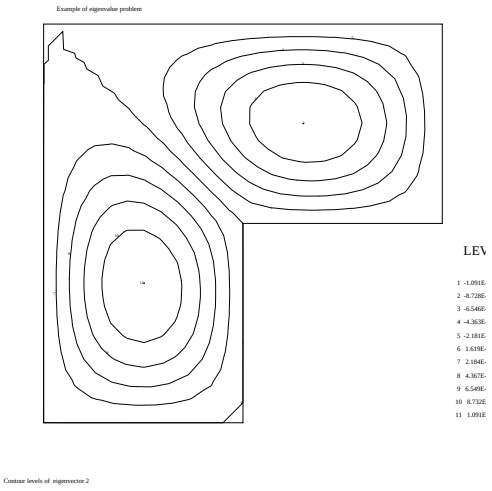


Figure 25.5.1.4: Contour plot of second eigenvector

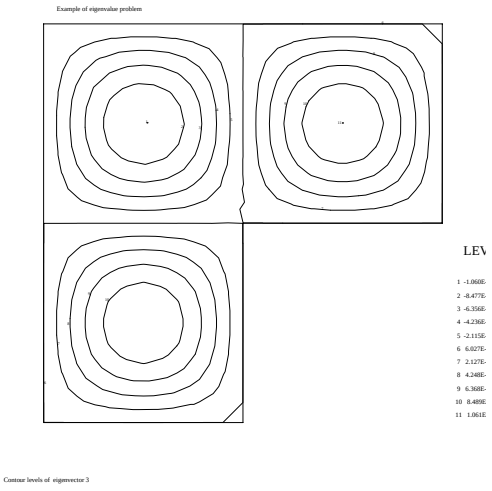


Figure 25.5.1.5: Contour plot of third eigenvector

References

Caswell and Viriyayuthakorn (1983) Finite element simulation of die swell for a Maxwell fluid, J. of Non-Newt. Fluid Mech. Vol 12, 13-29.

Cuthill, E. and J. McKee (1969) Reducing the band width of sparse symmetric matrices. Proc. ACM Nat. Conf. Association of Computing Machinery, New York.

Hasbani, Y. and M. Engelman (1979) Out of core solution of linear equations with non-symmetric coefficient matrix. Computer and Fluids, 7, p. 13-31.

Sloan, S.W. (1986) An algorithm for profile and wavefront reduction of sparse matrices. Int. J. for Num. Meth. in Engng, 23, p. 239-251.

Index