SEPRAN

SEPRA ANALYSIS

INTRODUCTION

GUUS SEGAL

Handleiding SEPRAN voor de practica op de TUD

September 1999

Ingenieursbureau SEPRA
Park Nabij 3
2267 AX Leidschendam
The Netherlands
Tel. 31 - 70 3871624

Fax. 31 - 70 3871943

Contents

1 Introduction

The aim of this introductory manual is to make it possible for an inexperienced user to run simple SEPRAN programs without the necessity of studying all the possibilities SEPRAN provides.
Chapter 2 gives some general remarks and definitions and should be read before studying the rest of this manual. The global structure of a SEPRAN-session is described in Chapter 3. Chapter 4 gives an introduction to the mesh generation, the computational part is treated in Chapter 5 and Chapter 6 is devoted to postprocessing. Finally Chapter 7 gives some examples of SEPRAN runs, including input and output.

How to use this manual?

In order to get a quick start it is recommended to proceed as follows:

- Read chapters 2 and 3 globally to get familiar with the notations used.
- Read Chapter 4 until Section 4.4
- Next read Chapter 5 until Section 5.4
- Now proceed by considering the examples in Chapter 7. Try to understand these examples by looking into previous chapters. The index may be used as a guide to find all is needed.

The complete set of SEPRAN manuals can be found in// `http://ta.twi.tudelft.nl/sepran`

2 General remarks and definitions

In this chapter we introduce some concepts and definitions that will be used throughout this manual. Consider the heat conduction problem as sketched in Figure 2.1 as an example.

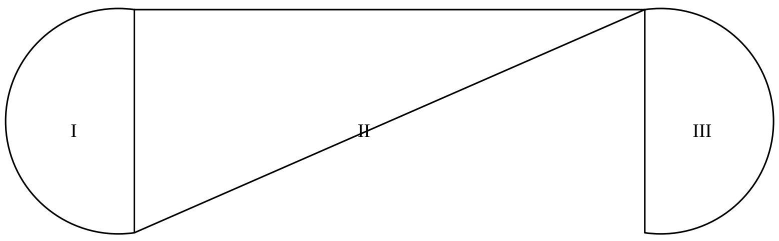


Figure 2.1: Region of definition for heat conduction problem with 3 subregions

The region defined by Ω consists of 3 subregions with heat capacities c_1, c_2, and c_3 respectively. The boundary conditions are defined in Figure 2.2.

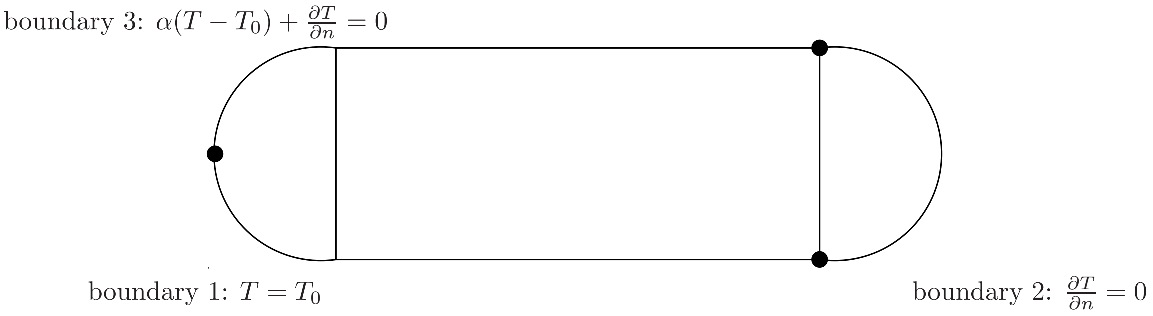

boundary 3: $\alpha(T-T_0)+\frac{\partial T}{\partial n}=0$
boundary 1: $T=T_0$
boundary 2: $\frac{\partial T}{\partial n}=0$

Figure 2.2: Definition of boundary conditions for heat conduction problem

boundary 1: Temperature is equal to outer temperature: T_0
boundary 2: No heat flow (isolated): $\frac{\partial T}{\partial n}=0$
boundary 3: Linearized radiation condition: $\alpha(T-T_0)+\frac{\partial T}{\partial n}=0$

The boundary condition of type $T=T_0$ is a so-called essential boundary condition for this problem, the boundary conditions on boundaries 2 and 3 are natural boundary conditions. For each problem in the manual Standard Problems it is described whether a boundary condition is essential or natural. For a definition of these types of boundary conditions, the reader is referred to the programmers guide.

2.1 Elements

One of the main items of the finite element method is the division into elements, which is called the discretization of the area Ω. This discretization will generally be applied to each subregion separately. See Figure 2.3.

Such a partition into elements is called the mesh, the partition of a subregion is called submesh. An element division is made by a (sub)mesh generator. See Section 4.1.

The necessity of defining a subregion may be induced by the physical properties of the matter in Ω, e.g. material properties, but subdivision might also be necessitated by approximation and/or refinement requirements. The elements may be one, two or three dimensional. In SEPRAN various types of elements may be used and connected. This has been sketched in Figure 2.3 where triangles and quadrilaterals are connected.

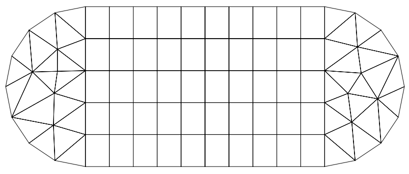


Figure 2.3: Division into elements (mesh), with 3 submeshes

All elements in a submesh must have the same shape, for example they are all triangles, quadrilaterals or line elements. (Line elements are one-dimensional elements in R^2 or R^3).
Furthermore all elements of a submesh have the same number of nodal points, which fixes the highest degree of approximation that can be performed with help of these elements. The (set of) differential equations need not be specified at this stage. The design of the mesh is a matter of geometry only, although the question of the necessity of local refinement is closely related to the problem to be solved.

The elements in a submesh are represented by one element, the so-called standard element. Later we shall see that a problem, a (set of) partial differential equations, can be described completely on a standard element. We then use the terminology: standard problem.
The standard element is not an element of the mesh, but gives the structure of the group of elements which is represented by it. For example, it defines the shape of the elements and the number of nodal points. In Figure 2.4 some examples of standard elements are plotted.

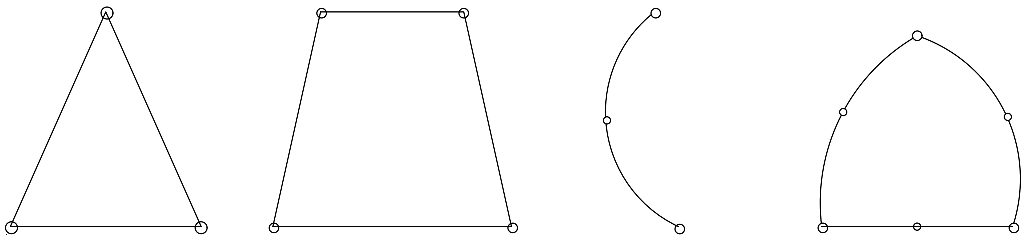


Figure 2.4: Some examples of standard elements

Each standard element is provided with a sequence number. When there are NELGRP standard elements, then the sequence numbers are IELGRP (IELGRP = 1 (1) NELGRP). Elements must be created with increasing sequence number. Hence first all elements with sequence number 1, then with sequence number 2 etc.

Elements in different submeshes may be represented by the same standard element. Consider the heat conduction problem of Figure 2.1 and suppose that the package contains standard problems defined on two different standard elements. The large system of equations (see Section 5.3) is built, starting with sequence number IELGRP=1,2,...,NELGRP. Therefore another value of the heat capacity c can be submitted for every submesh. Note that the submeshes may be represented by the same standard element, with different sequence numbers IELGRP=1,2,3..
When $c_1 = c_3$ the submeshes I and III may be created before submesh II, and then they are represented by a standard element with the same sequence number IELGRP=1.

2.2 Boundaries

If we consider the boundaries in Figure 2.2, we see that the physical boundary, that is the boundary of the region Ω, consists of 3 parts where different types of boundary conditions are given.
Consider the case in which the region of definition Ω is two-dimensional. In order to prescribe boundary conditions or compute special quantities a subdivision of the boundaries may be necessary. The physical boundary will be called outer boundary, it consists of one or more closed parts; each part can be divided into smaller pieces.

For example the outer boundary in Figure 2.2 consists of one closed part, the outer boundary in Figure 2.5 consists of 3 closed parts (I, II and III). The non-physical boundaries which have nothing but points in common with the physical boundaries are called inner boundaries. The closed parts of the outer boundaries start and end in the same point.
For three dimensional problems, see the programmers guide.
In plotting subroutines only the outer boundaries are used. Each boundary may be divided into one or more sub-boundaries.

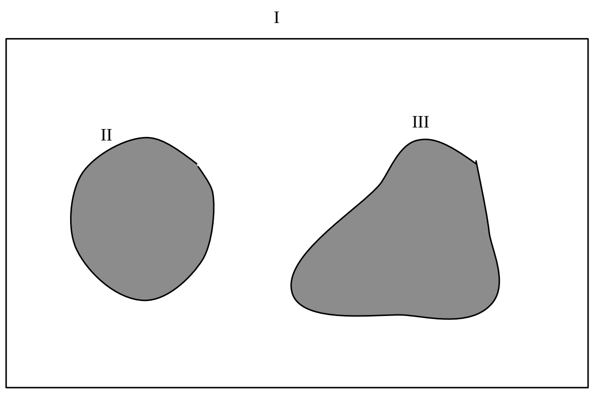


Figure 2.5: Region Ω containing 2 islands

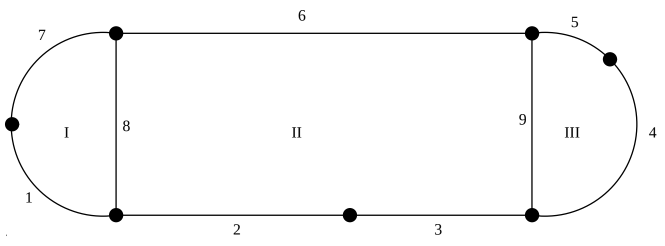


Figure 2.6: division into sub-boundaries

Boundaries are subdivided when the sub-boundaries are necessary for special reasons in the rest of the program, for example for the specification of boundary conditions.

For example for the region in Figure 2.2 the sub-boundaries are given in Figure 2.6. There are 7 outer boundaries and 2 inner boundaries. For the boundary conditions it is only necessary to define 3 outer boundaries: 1 + 2, 3 + 4 and 5 + 6 + 7. Whether the user defines 7 outer boundaries and 2 inner boundaries, or 3 outer boundaries only, is a matter of his own decision. The outer and inner boundaries are not necessary for the mesh generation.

Remark
In the case of a crack we define 2 outer boundaries with different boundary numbers. Although in that case the coordinates may coincide, each boundary is provided with separate nodal point numbers, except for the common node, at the tip of the crack. See Figure 2.9.

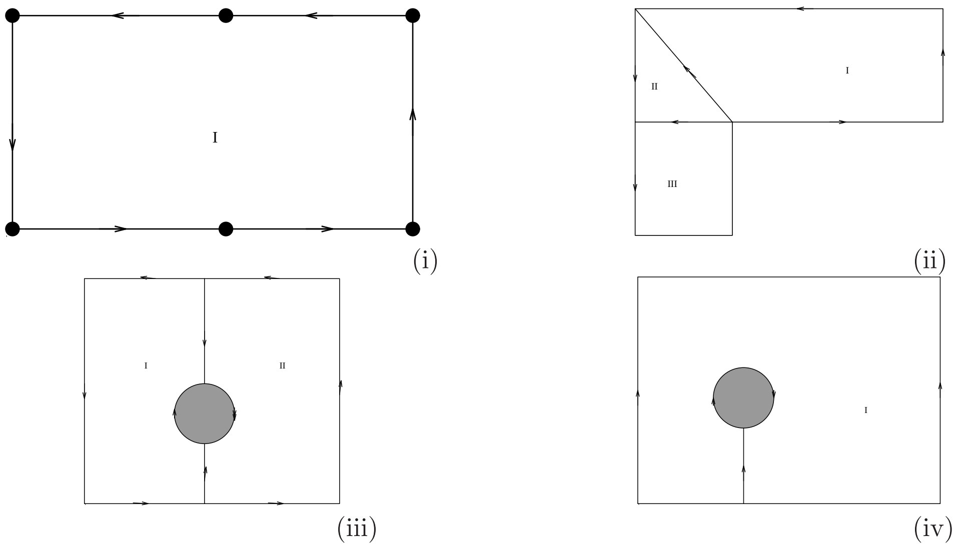


Figure 2.7: Examples of regions consisting of 1 (i),(iv) 3 (ii) and 2 (iii) subregions

2.3 Problem definition

Once the mesh has been generated, the problem that has to be solved can be defined. The physical or engineering problem consists of the set of partial differential equations (elasticity problem, flow problem and so on), and the boundary conditions.
In SEPRAN a standard problem is the result of the polynomial approximation of the unknowns of a special class of differential equations with respect to a standard element.
Each standard problem defined on a specific standard element is provided with a different problem definition number. This problem definition number must be connected with a standard element of the generated mesh, hence there are NELGRP problem definition numbers necessary.

The problem definition number completely defines the relation between the unknowns of the problem and the influence of physical constants and properties of the material. For example the standard problem defined by the number 800 means that the extended Convection-diffusion equation using an approximation defined in all nodes of the element is used.

Type numbers 1 to 99 may be used by the programmer to introduce his own standard problems, type numbers ≥ 100 are used for standard problems from the library of the package.

2.4 Boundary conditions

Boundary conditions can be of various type, depending on the equations to be solved. Boundary conditions must be prescribed on points, curves or surfaces (in R^3). It is only possible to create boundary conditions on complete curves and surfaces. Thus it is necessary to define these curves and surfaces adequately during the mesh generation. In this chapter we distinguish between the following possibilities:

2.4.1 Essential boundary conditions

In general, essential boundary conditions are the easiest to understand. Boundary conditions are called essential if they prescribe the value of some degrees of freedom. For example the boundary condition $T = T_0$ in Figure 2.2 is essential. So essential boundary conditions reduce the number of unknowns.

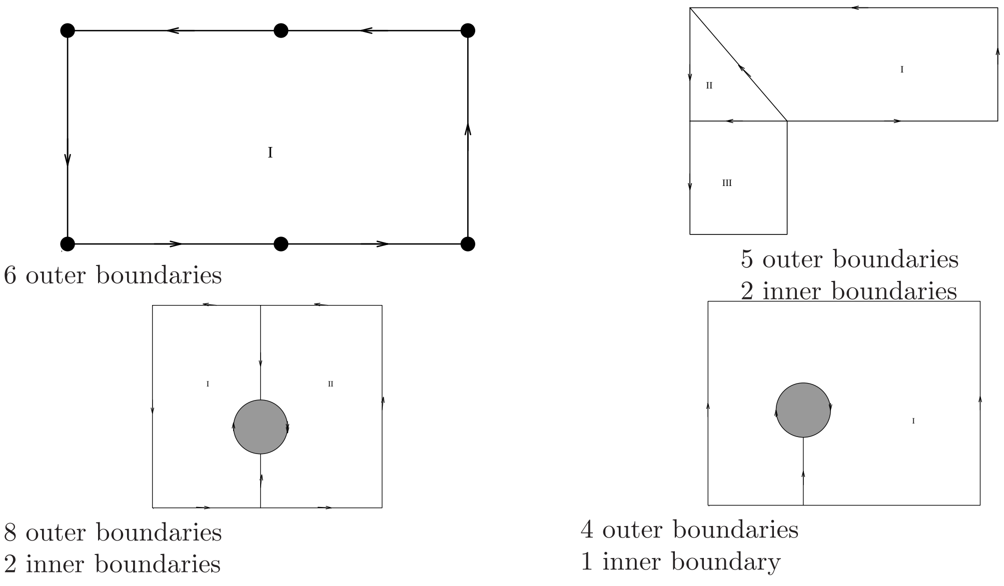


Figure 2.8: Examples of inner and outer boundaries each provided with a direction

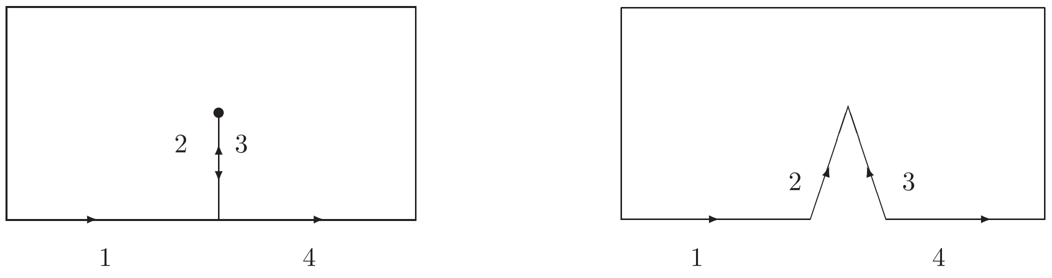


Figure 2.9: Treatment of a crack

2.4.2 Natural boundary conditions

Some boundary conditions give rise to surface (boundary) integrals to be evaluated. For example the radiation condition in Figure 2.2 gives rise to these boundary integrals. Instead of prescribing some degrees of freedom, the boundary conditions are "hidden" into these surface integrals. Such boundary conditions are called natural. In SEPRAN we have created so-called boundary elements in order to be able to calculate these surface integrals.

Boundary elements are defined without affecting the mesh, i.e. they are not treated as line elements. They are indicated by a standard boundary element sequence number, and a standard boundary problem definition number. Compare with the standard element sequence number and the standard problem definition number as defined in 2.3. Standard boundary elements are created in in a natural sequence, with standard boundary sequence numbers from 1 to NUMNATBND, with NUMNATBND the number of standard boundary element sequence numbers.

2.4.3 Periodical boundary conditions

In some problems periodical boundary conditions are prescribed on opposite boundaries. For example in Figure 2.10, the boundaries I and III may have periodical boundary conditions. In that case the corresponding degrees of freedom must be identified.

Therefore the user must introduce elements from boundary I to boundary III. These elements must be created by the mesh generator (SEPMESH). See "special purpose elements", under the heading

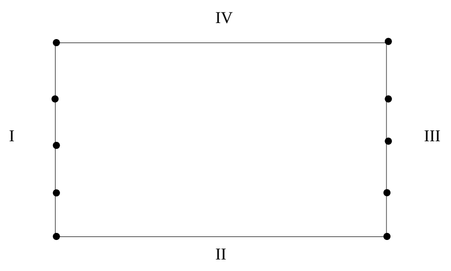


Figure 2.10: Periodical boundary conditions on sides I and III.

MESHCONNECT. These elements must get the type number -1 in the input block "PROBLEM", indicating that these elements have periodical boundary conditions. All degrees of freedom in nodal points of boundary I are identified with the corresponding degrees of freedom on boundary III. For the example of Figure 2.10 the elements created are sketched in Figure 2.11.

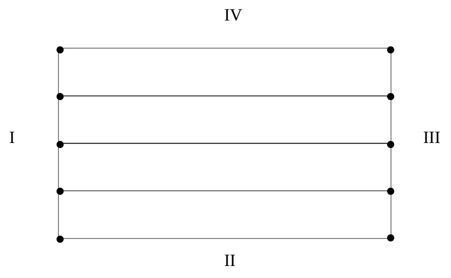


Figure 2.11: Elements to prescribe periodical boundary conditions on the sides I and II

2.5 Some special definitions

The following definitions are used throughout the SEPRAN-manuals:

Degrees of freedom: degrees of freedom refer to the unknowns. Sometimes the degrees of freedom refer to all unknowns in the mesh, in other cases only the unknowns in a nodal point are meant.

Prescribed degrees of freedom: Some boundary conditions explicitly prescribe the values of the unknowns. For example the temperature given in a part of the boundary is such a prescribed boundary condition. Boundary conditions of this type are called essential boundary conditions. The corresponding unknowns are referred to as prescribed degrees of freedom. The number of unknowns is reduced by these boundary values.

COMMAND: In the definition of the input for a number of subroutines the notion COMMAND is used. With COMMAND a part of the input for a specific subroutine or a part of a program is meant that is obligatory for that subroutine. DATA records on the other hand define only values and options and are usually optional. A COMMAND is always started in a new record in the input file.

3 The global structure of a SEPRAN-session

Except for complicated problems, the SEPRAN-session consists of three parts, which can be run separately. The three parts are

- Preprocessing (creation of mesh)
- Computation (definition and solution of the problem)
- Postprocessing (output of the results: plots and prints)

The sequence of the session is always:

preprocessing followed by computation followed by postprocessing

In the sections 3.1, 3.2, and 3.3, the various stages are treated separately.

3.1 The preprocessing part of SEPRAN

The preprocessing part of SEPRAN consists of the creation of the mesh. In this manual only one and two dimensional meshes are treated; for three dimensional regions the reader is referred to the Users Manual.

The generation of the mesh is performed by the program SEPMESH. At this moment only a batch version is available, which requires a file with data. This file must be created by the user for example with a text editor. An interactive version of SEPMESH is under development.

Program SEPMESH creates output in two ways:

- SEPMESH writes to the standard output device (usually the display from which you start the program, or a standard output file). This output consists of a copy of the input file, error messages if the input is incorrect and some messages from sub mesh generators.
- If the input is error-free and a mesh has been generated, then this mesh is written to a file named meshoutput.

Chapter 4 describes how a 2D-mesh may be generated, for a 3D-mesh the user is referred to the Users Manual.

SEPMESH must be used as follows:

```
sepmesh  inputfile
```

or

```
sepmesh  inputfile  >  outputfile
```

The inputfile is the file created by the user using the text-editor. If no outputfile is specified all information (including error messages) is written directly to the screen.
The output file may have any name except meshoutput and sepplot.$* * *$, where $*$ is any number.

Example: sepmesh mesh.dat > mesh.out. (unix)

Remark: besides the file meshoutput sepmesh also creates files sepplot.001, sepplot.002, etc. which contain plot information.

So sepmesh creates output in 3 ways:

- a file meshoutput containing the complete description of the mesh;
- files sepplot.001, sepplot.002, etc. containing information of the plots to be made;
- output written to the screen or the outputfile (for example mesh.out). This output contains a hard copy of the input, error messages (if any) and some information about the mesh.

In Section 3.4 it is described how the plot information may be translated into a plot.

3.2 The computational part of SEPRAN

In the computational part of SEPRAN first the mesh created by the preprocessing part is read, then the type of problem is defined, the system of equations is built (including boundary conditions) and finally the problem is solved. The result of this part is written to a file which can be used at the output part. SEPRAN is developed to solve very complicated problems. However, in this introduction only very simple standard problems are treated, which require only a minimum of input.

For simple problems SEPRAN provides a standard program: SEPCOMP. If the problem to be solved fits within the frame-work of SEPCOMP, there is no need to create a main program. In that case it is sufficient to run SEPCOMP itself. The input required for SEPCOMP is described in Chapter 5.

If the program SEPCOMP does not offer all the possibilities, that are needed for the solution of a specific problem, it is necessary to write a main program. SEPRAN provides a large number of subroutines in an increasing sequence of detail, which can be used to construct such a main program. The main subroutines are treated in the SEPRAN users manual, for an extended description the reader is referred to the programmers guide.

The computational part of a simple problem consists of the following components:

- In the starting part the mesh is read from the file created by SEPMESH, the definition of the problem, and the type of solver is read.
- In the next phase the essential boundary conditions are read, i.e. the prescribed unknowns.
- In the third phase the coefficients (material properties) are read, the system of equations is built and the problem is solved.
- Finally some derived quantities (like for example gradient, pressure, stream function) may be computed, and the solution as well as these derived quantities are written to a file.

SEPCOMP must be used as follows:

```
sepcomp  inputfile
```

or

```
sepcomp  inputfile  >  outputfile
```

The inputfile is the file created by the user using the text-editor. If no outputfile is specified all information (including error messages) is written directly to the screen.
Besides the user provided inputfile, sepcomp also needs the file meshoutput created by sepmesh.
The outputfile may have any name except meshoutput, sepcomp.inf, sepcomp.out and sepplot.$***$, where $*$ is any number.

Example: sepcomp comp.dat > comp.out.

Remark: sepcomp creates two files sepcomp.inf and sepcomp.out, which contain information for the postprocessing.

So sepcomp creates output in 2 ways:

- two files sepcomp.inf and sepcomp.out containing the complete description of the solution computed;
- output written to the screen or the outputfile (for example comp.out). This output contains a hard copy of the input, error messages (if any) and some information about the problem solved.

3.3 The postprocessing part of SEPRAN

In the postprocessing part of SEPRAN, the mesh is read as well as the solution, as created by the computational part. In this section the results are produced for the user in a more suitable form: prints, plots, integrals etc. The postprocessing part is performed by the program SEPPOST. For a description of its possibilities the reader is referred to Chapter 6.

SEPPOST is used in the same way as SEPMESH, i.e. the user creates an input file by the text-editor and then runs SEPPOST.
SEPPOST is used as follows:

```
   seppost  inputfile
```

or

```
   seppost  inputfile  >  outputfile
```

The inputfile is the file created by the user using the text-editor. If no outputfile is specified all information (including error messages) is written directly to the screen.
Besides the inputfile SEPPOST uses also the files created by SEPMESH (meshoutput) and the SEPRAN computational program (sepcomp.inf and sepcomp.out).
SEPPOST does not produce plots directly but produces files named sepplot.001, sepplot.002, etc. containing plot information. Since this name is the same as for SEPMESH the plot information of SEPMESH is destroyed. To display the plot exactly the same procedure as for SEPMESH must be used.

Example:

```
   seppost  post.dat > post.out
```

3.4 Display of SEPRAN plots

The SEPRAN mesh generation part or the postprocessing part may generate plot files named sepplot.001, sepplot.002, sepplot.003, etc.

In SEPRAN there are two ways of displaying these plots: you can make a picture at the screen, or you make a plot onto a laser printer or plotter.

To display the plot on the screen use the command:

```
sepview
```

or

```
sepview sepplot.xxx
```

where sepplot.xxx is the file to be plotted.
If sepview is used without file name, the file may be selected by the option file. Once a file is selected all files with the same basename and extension .001, .002, ... may be viewed. The first file is the file selected.
SEPVIEW has the following options:

Zooming in Press the left mouse button down and move the cursor upwards while pressing the button. Release the mouse button if the created rectangle is large enough. The picture within the rectangle will be drawn in the full window.

Zooming out Zooming out means displaying the previous window. Zooming out is done by moving the cursor downwards while creating a rectangle.

Panning Panning is done by pushing and releasing the left mouse button on the same place in the picture. The picture is panned towards the mouse position. How much the picture is panned depends on the distance between the mouse button and the middle of the picture.

Hardcopy Pressing the 'Hardcopy' button will show you a pull down menu with several possible choices. Press 'Postscript' to produce an encapsulated postscript file <sepplot.xxx>_<nn>.eps from the current view. <nn> is a sequence number. It will be increased each time a new file is generated.
Once you have left sepview, you may print the files `sepplot.001_01.eps ...` on the laser printer. The command to be used is:

```
laserps sepplot.001_01.eps
```

where the name `sepplot.001_01.eps` must be replaced by the one to be printed.

Play / Stop / Previous / Next At the upper right corner of the plot window, there are three buttons, a left arrow, a right arrow and a push button labelled 'Play' or 'Stop'.

The name of a SEPRAN plot file is of the form sepplot.xxx, where xxx is a number. This number can be used to select a previous/next plot file of the same set with a higher/lower number, using the right and left arrow. If there is no plot file with a higher/lower number, the right/left arrow is disabled.
To show a set of plot files as an animation, you can use the 'Play' button. As soon as SEPVIEW has started playing, the label on the button is changed to 'Stop' to stop the animation. As soon as the last file in the set is shown, the animation is reversed.

3.5 An overview of the simple SEPRAN commands

In this section we give an overview of some of the available SEPRAN commands.

The following SEPRAN commands are available:

sepmesh (creates a SEPRAN mesh, see Section 3.1)

sepcomp (performs the computational part of SEPRAN, see Section 3.2)

seppost (performs the SEPRAN postprocessing, see Section 3.3)

sepview (Plot SEPRAN files under X, see Section 3.4)

seplink (Link a SEPRAN main program and subroutines with the SEPRAN libraries, see Section 5.2)

4 Mesh generation

4.1 General remarks

Before reading this section the user should consult Section 2 for some definitions.

The generation of submeshes may be done by a standard submesh generator, by a user written submesh generator or by input from the standard input file. In this manual only the first possibility is treated, for the other cases see the Users Manual. Furthermore this manual is restricted to one- and two-dimensional meshes only.

The definition of the elements is performed in two stages:

- in the first stage the user defines geometrical quantities as points, curves, surfaces and volumes, and elements along these quantities,
- in the second stage elements created in the first stage are coupled to element groups. Only those elements necessary for the solution of the finite element problem must be identified with an element group.

4.1.1 Definition of points, curves, surfaces and volumes

For the generation of meshes we define the following quantities:

Points, Curves, Surfaces and Volumes

Points form the basis for all other components. The user must define the main points necessary for the generation of curves. These points must be numbered sequentially from 1 onwards. After the generation of the mesh they are connected to nodal point numbers. The corresponding nodal point numbers are generally not equal to the point numbers defined by the user.

Curves form the one-dimensional quantities of the meshes. For example lines and arcs are curves. The initial and end points of any curve must already have been defined as points. Curves have an orientation, defined by the initial and end points, hence line C3 = (P3, P4) is different from line C4 = (P4, P3).

Surfaces form the two-dimensional quantities of the mesh. The boundaries of the surfaces must already have been defined as curves. The boundary of a surface must be closed in itself, the internal part of the surface must be on the left-hand side of the curves. Hence the boundaries of a surface must be created counter clockwise and may not intersect itself. Whenever in a description of a surface a curve is needed in the opposite direction of which it was defined, then its number must be preceded by a minus sign. (See Figure 4.1.1). For "exotic" boundaries it may be wise to divide the region considered in a number of less "exotic" subregions since most of the SEPRAN generators will give better results in such a situation and besides also require less computation time.

Volumes form the three-dimensional quantities of the mesh. They are not defined in this manual. For three-dimensional problems the user is referred to the Users Manual.

All points, curves, surfaces and volumes must be numbered sequentially, each starting with number one. The outer and inner boundaries as defined in 2.2 must consist of points (in R^1), points and curves (in R^2), and points, curves and surfaces (in R^3).

The submeshes as defined in 2 must coincide with curves (in R^1), surfaces and sometimes curves (in R^2), or with volumes and sometimes curves and surfaces (in R^3).

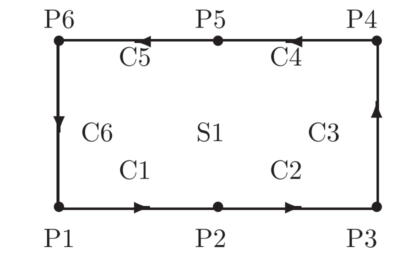
Anywhere in the manuals where curves, points and surfaces are mentioned, the curves,

points and surfaces generated by the mesh generator are meant. Nodal points of the mesh must be coupled with these points, curves and surfaces.

Examples

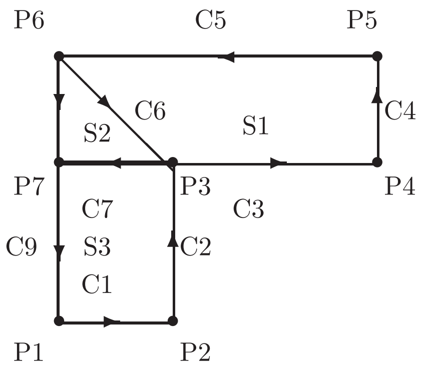
Consider the regions in Figure 2.7 and 2.8. In Figure 4.1.1 the points, curves and surfaces for these regions are defined. Points are indicated by Pk (k=1 ,2 ,,,), curves by Cl (l=1 ,2 ,,,) and surfaces by Sm (m=1 ,2 ,,,). The corresponding commands are POINTS, CURVES and SURFACES. *Remark:*

When the user wants to create double points on a line (for example for a crack), he has to introduce two curves with the same end points on this line. For example the outer boundaries 2 and 3 in Figure 2.9 must be created as 2 different curves.



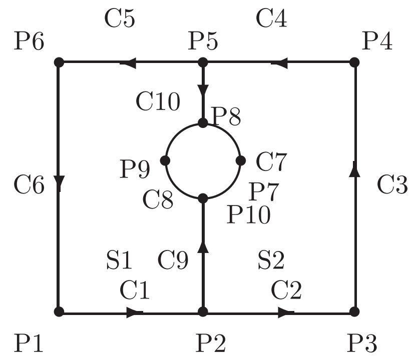
C1 = (P1,P2) C2 = (P2,P3)
C3 = (P3,P4) C4 = (P4,P5)
C5 = (P5,P6) C6 = (P6,P1)
S1:(C1,C2,C3,C4,C5,C6)

Outer boundaries: C1, C2, C3, C4, C5, C6



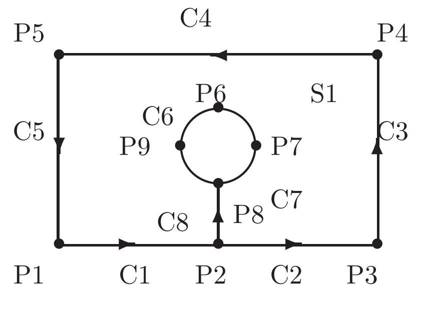
C1 = (P1,P2) C2 = (P2,P3)
C3 = (P3,P4) C4 = (P4,P5)
C5 = (P5,P6) C6 = (P6,P3)
C7 = (P3,P7) C8 = (P6,P7)
C9 = (P7,P1)
S1:(C3,C4,C5,C6)
S2:(-C7,-C6,C8)
S3:(C1,C2,C7,C9)

Outer boundaries: C1, C2, C3, C4, C5, C8, C9
Inner boundaries: C6, C7



C1 = (P1,P2) C2 = (P2,P3)
C3 = (P3,P4) C4 = (P4,P5)
C5 = (P5,P6) C6 = (P6,P1)
C7 = (P8,P7,P10) C8 = (P10,P9,P8)
C9 = (P2,P10) C10 = (P5,P8)
S1:(C1,C9,C8,-C10,C5,C6)
S2:(C2,C3,C4,C10,C7,-C9)

Outer boundaries part 1: C1, C2, C3, C4, C5, C6
Outer boundaries part 2: C7,C8
Inner boundaries: C9, C10



C1 = (P1,P2) C2 = (P2,P3)
C3 = (P3,P4) C4 = (P4,P5)
C5 = (P5,P1) C6 = (P6,P9,P8)
C7 = (P8,P7,P6) C8 = (P2,P8)
S1:(C8,-C6,-C7,-C8,C2,C3,C4,C5,C1)

Outer boundaries part 1: C1, C2, C3, C4, C5
Outer boundaries part 2: -C6, -C7
Inner boundaries: C8

Figure 4.1.1: Points, Curves and Surfaces

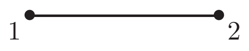
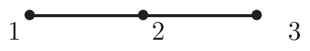
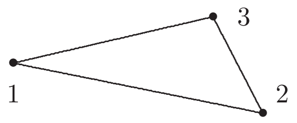
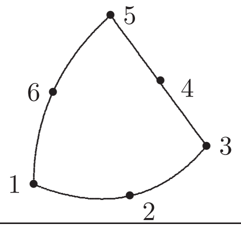
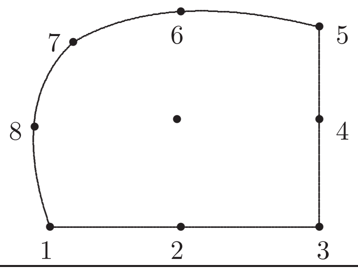
shape number	shape	name
1		line element with 2 points
2		line element with 3 points
3		triangle with 3 points
4		isoparametric triangle with 6 points
5		quadrilateral with 4 points
6		isoparametric quadrilateral with 9 points

Table 4.1.1: Standard elements for mesh generation

4.1.2 Generation of curves

First the user must define the points, secondly the curves and finally the surfaces.
For the definition of the curves the user may specify the number of nodal points on a curve as well as the distribution of these points. Another possibility is to define an approximate length of the elements in the end points of the curves. Elements in between are defined such that the mesh size increases or decreases monotone and smoothly from one end to the other. When the user wants to utilize this possibility he must give the command COARSE, and give a unit length (UNIT). Furthermore each user point must be provided with a so-called coarseness (c). Then the approximate length of the elements in the surroundings of these points is equal to $c\times$ UNIT, depending on the type of function that is used for the creation of the curve.
For the definition of the curves the following FUNCTIONS are available:

LINE $<$ *element_type* $>$: generates a straight line from point Pi to Pj.

ARC $<$ *element_type* $>$: generates an arc from point Pi to Pj; the centroid Pc must be given.

USER $<$ *element_type* $>$: the user gives all coordinates of the nodal points on the line.

CLINE $<$ *element_type* $>$: generates a straight line from point Pi to Pj, where the elements are defined with the concept of coarsenesses.

CARC $<$ *element_type* $>$: generates an arc from point Pi to Pj; the centroid Pc must be given. The elements are defined with the concept of coarsenesses.

CURVES : generates a curve consisting of the subsequent curves Ck, Cl, Cm.

PARAM The user defines a curve by a function subroutine FUNCCV (4.4.1) using a parameter representation.

CPARAM The user defines a curve by a function subroutine FUNCCV (4.4.1) using a parameter representation. The division of elements is based on the concept of coarseness.

For other functions the reader is referred to the Users Manual. $<$ *element_type* $>$ is an integer which defines the type of elements along the curves to be created.

The FUNCTIONS LINE, ARC, USER, SPLINE, CLINE, CARC, CURVES, PARAM and CPARAM have the following shape:

```
C1 = LINE <element_type> ( P1, P2, NELM = n, RATIO = r, FACTOR = f )
C2 = ARC  <element_type> ( P1, P2, Pc, NELM = n, RATIO = r, FACTOR = f )
C3 = USER <element_type> ( P1, P2, P3, . . . , Pn )
C4 = CLINE<element_type> ( P1, P2, NODD = o )
C5 = CARC <element_type> ( P1, P2, Pc, NODD = o )
C6 = CURVES ( Ck, Cl, Cm, . . )
C7 = PARAM <element_type> ( P1, P2, NELM=n [,INIT=t_0] [,END=t_1] //
                           [, RATIO=r, FACTOR=f ] )
C8i = CPARAM <element_type> ( P1, P2 [,NODD=o [,INIT=t_0] [,END=t_1])
```

with n the number of elements in the curve.

The distribution of the nodal points is given by the parameters RATIO and FACTOR:

r=0:	equidistant mesh size (default)
r=1:	the last element is f times the first element
r=2:	each consecutive element is f times the preceding element

The value of o defines whether the number of end points of the elements on the curve is free, odd or even. Possibilities:

o=0,1	free
o=2	number of end points odd
o=3	number of end points even

$< element\_type >= 1$ means linear elements, consisting of 2 points (Default value)
$< element\_type >= 2$ means quadratic elements, consisting of 3 points, with the second point in the centre of the first and the last one.
INIT = t_0 and END = t_1, define the range of the parameter t. The default values are: $t_0 = 0$ and $t_1 = 1$

4.1.3 Generation of surfaces

Each surface must coincide with a submesh (in two-dimensional problems). For generation of nodal points and elements in the surface a number of so-called surface generators are available. Of these surface generators only two are treated in this manual. For the other ones the user is referred to the Users Manual.

The surface generators described in this manual are GENERAL and QUADRILATERAL.

GENERAL has the following characteristics:

1. A fine division of nodal points on a part of the boundary causes a fine mesh in the neighbourhood of this boundary; a coarse division, a coarse mesh.
2. The mesh generator can not generate elements when a sudden refinement of the nodal points of the boundary is present. Hence when the user wants to create elements on a long small pipe (see Figure 4.1.2) GENERAL can not be used, or the user must transform his coordinates such that the length/width ratio is not too large. For that type of meshes use QUADRILATERAL.
3. If the boundary is too random (Christmas tree), a subdivision into submeshes may be necessary.
4. When quadrilateral elements are generated by GENERAL it is necessary that the number of nodal points on the boundary of the surface is even in the case of linear elements, and the number of vertices of elements on the boundary is even in the case of quadratic elements. The user must take care of this demand.

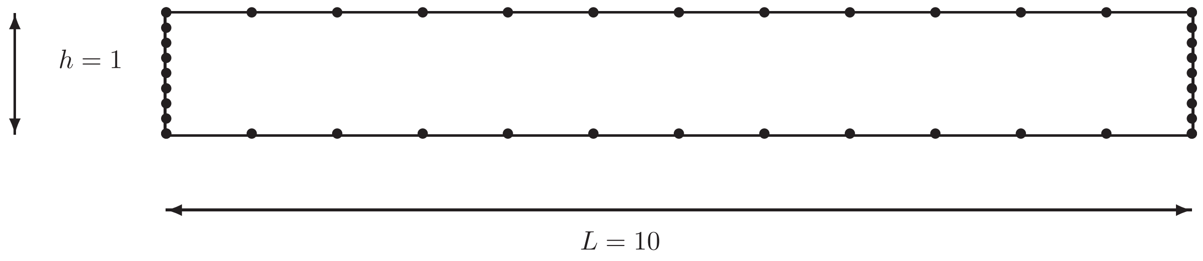


Figure 4.1.2: Example of a region that can not be subdivided by GENERAL

For some examples of meshes created by GENERAL see Figure 4.1.3

QUADRILATERAL has the following characteristics:

1. The submesh generator QUADRILATERAL creates a mesh for regions that can be mapped onto a rectangle. Besides that, the region must be topological equivalent to a rectangle. Topological equivalent to a rectangle means that a mapping onto a rectangle must be possible. The sides of the region may be curved, but the curvature may not be so extreme that there is no resemblance with a rectangle.
2. QUADRILATERAL expects exactly four curves, each one representing one "side" of the transformed "rectangle". If some of these sides consist of subcurves the user must combine these curves into one curve using the option CURVES (of curves).
3. When quadrilaterals are required the number of points on the four curves together has to be even. The user has to take care of this himself.
4. QUADRILATERAL has no problem with oblong elements.

For some examples of meshes created by QUADRILATERAL see Figure 4.1.4.

The functions GENERAL and QUADRILATERAL have the following shape:

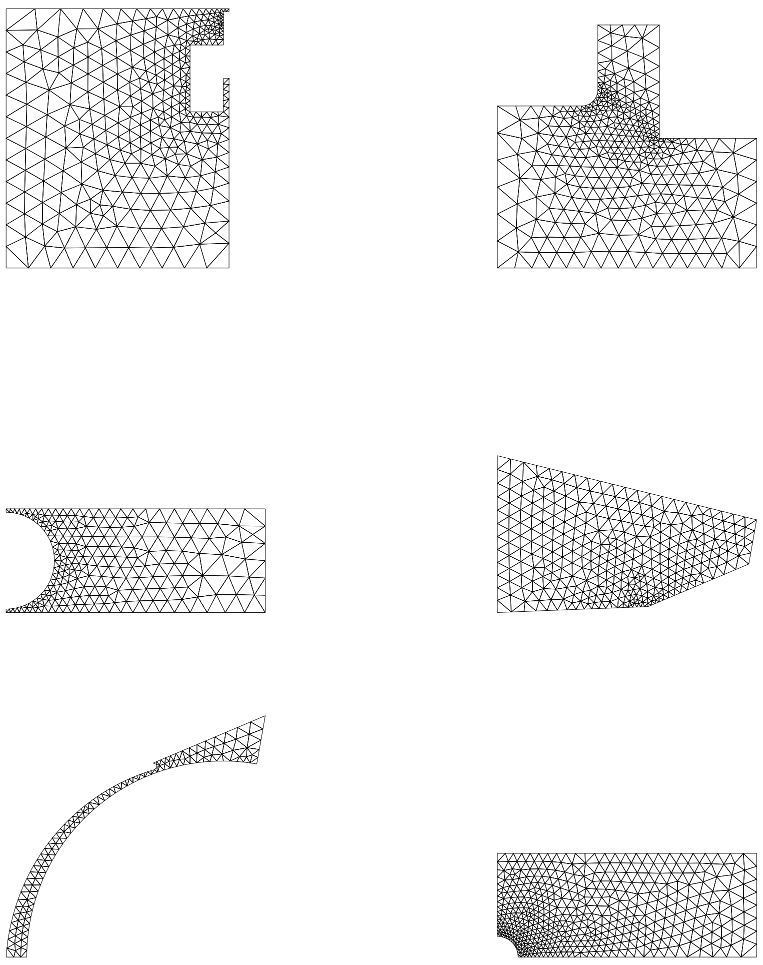


Figure 4.1.3: Example of meshes that can be created by GENERAL

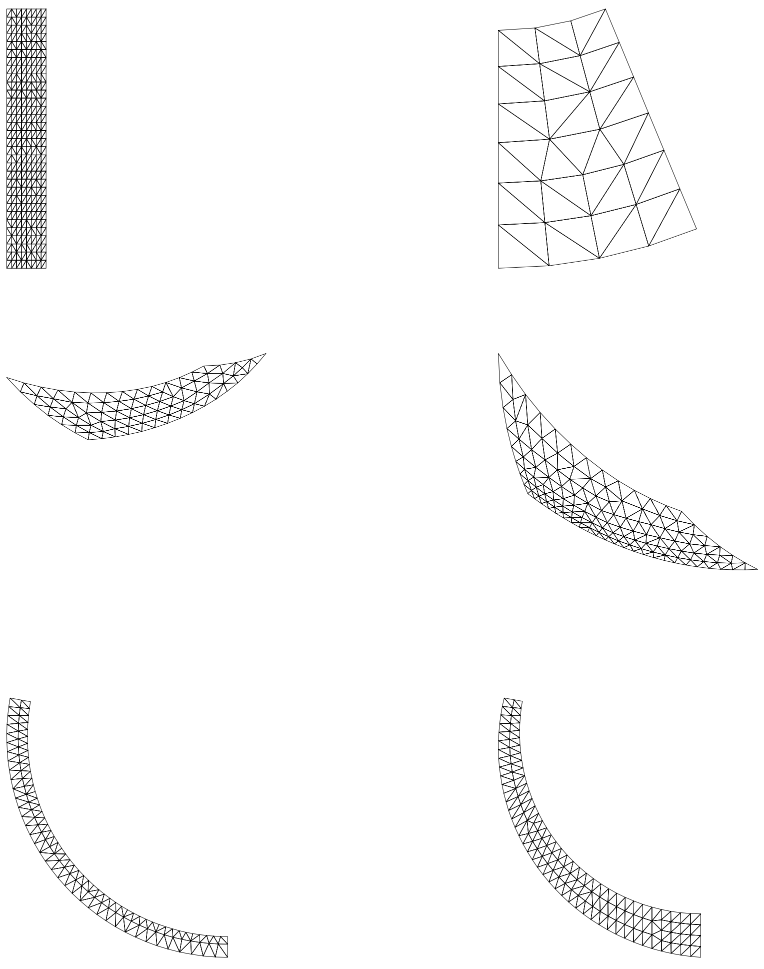


Figure 4.1.4: Example of meshes that can be created by QUADRILATERAL

```
S1 = GENERAL <element_type> ( C1, C2, C3, C4 . . . )
S2 = QUADRILATERAL <element_type> ( C1, C2, C3, C4)
```

$< element\_type >$ is an integer which defines the type of elements in the surfaces to be created. In Table 4.1.1 a survey of the available standard elements for mesh generators is given. The element types 3 to 6 can be generated by GENERAL and QUADRILATERAL. In the manual Standard Problems it is described which type of elements are available for a specific problem.

4.1.4 Coupling of geometrical quantities with element groups

The points, curves and surfaces as defined in 4.1.1 to 4.1.3 are necessary to generate elements. However, not all of them may be necessary for the finite element problem. Those elements that are necessary must be identified with a standard element by means of an element group number (see 2.1).
The coupling of the geometrical elements with the standard elements is done using the commands MESHLINES and MESHSURFACES defining the one and two dimensional elements respectively. These commands must be followed by function cards of the type:

```
LELM i = ( SHAPE = j, C1 ,C2)
```

or

```
SELM i = ( S1, S2 )
```

with `LELM` corresponding to the line elements and `SELM` corresponding to the surface elements.
i is the element group number.

For line elements the shape number for the generation of the elements must be given. This number gives the number of nodal points in an element minus one, hence j = 1: linear element, j = 2: quadratic etc. This number does not have to be identical to the shape number in the curve generation. The line elements are created on the curves C1 to C2, in that sequence.

The shape number of the surface elements is the same as the number corresponding to the surface elements S1 to S2, and hence must not be given in the function.

4.2 A simple example

Before we describe the input for the mesh generator in detail we shall first give an example to show how a simple mesh may be created.
To that end we consider a rectangular region as sketched in Figure 4.2.1. The user points and

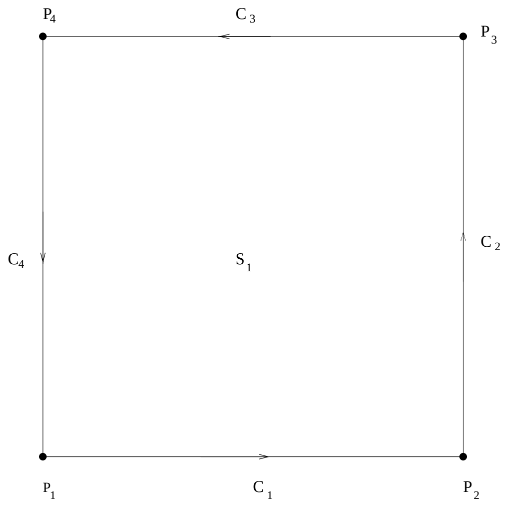


Figure 4.2.1: Example of a region to be divided in elements

curves are indicated in the region. Suppose that the height is 1 and the width is also 1.
In order to create a mesh by SEPRAN we first have to make an input file by a text editor. Suppose this input file is called `square.msh`.
In order to create the mesh we have to call the program sepmesh in the following way:

```
sepmesh square.msh
```

If the input file is incorrect, sepmesh produces error messages, which are self-explaining. Sometimes, however, the number of errors is so large that more than one screen is needed. In that case it might be wise to redirect the file to an output file, for example:

```
sepmesh square.msh > square.out
```

Never use the name `meshoutput` for this output file.
The output file may be inspected by a text editor.
Then sepmesh creates a mesh and puts information in a file called `meshoutput`. Depending on the contents of the file square.msh a series of files `sepplot.001`, `sepplot.002` ... may be created which contain plots related to the mesh. These plots may be viewed by one of the SEPRAN display programs like `sepview`. See Section 3.4.
The input file square.msh may for example look like:

```
#
#  square.msh
#
#  Example file for the SEPRAN introduction, Section 4.2
#
#  Define some constants for the mesh, See introduction, Section 4.3
#
constants
   reals
      height = 1                    # height of the square
      width  = 1                    # width of the square
   integers
      nelm_hor  = 10                # number of elements in horizontal direction
      nelm_vert = 10                # number of elements in vertical direction
end
#
#    Actual definition of the mesh
#
mesh2d

   #   Definition of the coordinates the user points

   points
      p1=(0,0)
      p2=($width,0)
      p3=($width,$height)
      p4=(0,$height)

   #   Definition of the curves

   curves
      c1=line(p1,p2,nelm=$nelm_hor)
      c2=line(p2,p3,nelm=$nelm_vert)
      c3=line(p3,p4,nelm=$nelm_hor)
      c4=line(p4,p1,nelm=$nelm_vert)

   #   Definition of the surface

   surfaces
      s1=general3(c1,c2,c3,c4)

   #   Plot the mesh

   plot
end
```

Explanation:

- In the part `constants ... end`, some general constants with respect to the mesh are defined. In this case, the width and the height of the mesh and the number of elements in horizontal and vertical direction. In this way it is easy to change these numbers later on.
- The part `mesh2d ... end` is meant for the actual mesh generation.
  First all user points are defined, next the curves as straight lines with linear elements (`line1`), begin point, and point and number of elements.
  After that, the surface is defined using the submesh generator general with triangular elements (`general3`), and corresponding curves and finally a plot command is given.
  The dollar-sign before the constants in the mesh definition, indicate that the values as defined in the constants part must be used.
  Everything after the hash symbol is treated as comment.

4.3 Some remarks concerning the input files

In the previous section we have seen an example of a simple input file. In the next section we shall treat a part of the input for the mesh generator. But before doing so we consider some general rules that are valid for all SEPRAN input files that are defined in the SEPRAN manuals, unless otherwise stated.
First of all it must be noted that the input file is not a FORTRAN file, hence rules that apply for FORTRAN files are not generally applicable to the SEPRAN input files. In fact each input file is interpreted, character for character.
The following rules are generally applicable for the input files:

- At most 80 characters in each line of the input file are read, all characters that are present after column 80 are neglected.
- If an input line requires more than 80 columns continuation of this line may be defined by putting the characters `//` after the last text on a line, but of course within the columns 1 to 80. This means that the line is continued on the next line.
  For example the next three lines are considered as one line:

  ```
  c4 = line1 ( p1, p2,//
               nelm = $nelm_hor //
               ratio = 3, factor = 0.5 )
  ```

- You may put comment in the input file in two ways:
  1. By putting a `*` in column 1. The whole line is treated as comment.
  2. By putting a hash (`#`) in the text. All characters behind the hash are treated as comment.
- SEPRAN does not distinguish between capitals and lower case, except in character strings.
- Numbers must satisfy the standard FORTRAN rules. However, they may not contain spaces.
  Examples are `1   1.0    1d0 1.0d0 1e0 1.0e0 0.01 .01 -0.01`
- Spaces and end of lines are treated as separation symbols. Also special characters as `, = : ;` may be used as separator.
- It is not possible to use special characters like `+ - *` to define an expression in the input file. Hence an expression like `3*4` is not recognized and must be replaced by 12.
- The input file may start with a part `CONSTANTS` to define some general constants. This part has the following layout:

  ```
  CONSTANTS
     INTEGERS
        1: name = value
        3: name
           name = value
     REALS
        1: name = value
        3: name = value
           name = value
     VARIABLES
        1: name = value
        3: name = value
           name
     VECTOR_NAMES
        1: name
        2: name
  END
  ```

These records have the following meaning

CONSTANTS (mandatory). This keyword indicates that constants will be defined.
If this keyword is not present as first keyword in the file it is not possible to define constants. This keyword may be followed by the subkeywords (always on a new line):

INTEGERS This keyword indicates that some integer constants will be defined.
It must be followed by the integers to be defined.
The layout of the integers is:

```
name_of_constant value
```

name_of_constant defines the name of the constant. The name must start with a letter and may consist of letters, digits and underscore signs only. All other signs are treated as separation sign, including the blank space. The name of the constant may be used in the rest of the input file as reference to the constant. This reference must be preceded by the $-sign, to indicate that it is a reference and not a keyword.

value must be a number according to standard FORTRAN rules. Spaces in the number are treated as separation character. If value is given the constant gets an initial value.

REALS This keyword indicates that some real constants will be defined.
It must be followed by the reals to be defined according to exactly the same rules as for the integers. Names of reals must be different from the names of the integers.

VARIABLES This keyword indicates that some variables will be defined.
It must be followed by the variables to be defined according to exactly the same rules as for the integers.
The difference between a variable and a real or integer constant is the following:
Constants that are used in the input file will be interpreted at the moment they are read. Then the value of the constant is substituted instead of the name of the constant. The reference to the constant must always be $-sign immediately followed by the name of the constant (no spaces allowed). Hence if the constant changes later on this has no effect anymore.
Since all input is read at the start of the input, this means that there is hardly any possibility to change the constant.

On the other hand variables are connected to the scalars as defined in the input file. See Section 5.4.11. A reference to a variable must be preceded by a %-sign immediately followed by the name of the scalar.
Scalars are evaluated at the moment they are used and may be recomputed during the execution of the program. Hence they allow a larger flexibility to manipulate. Internally this means that the value of the variable is not substituted during reading, but that a reference to the variable is made.
At this moment variables can only be used in combination with the keyword STRUCTURE as defined in Section 5.4.11.

VECTOR_NAMES This part makes only sense for the computational part and for the postprocessing. It offers the possiblity to give an output vector a specific name. This name may be used instead of a vector number, in order to increase readability. The reference to vectors defined in this block must be preceded by a %-sign. The input in this block is number followed by name, each on a new line. The solution vector is always coupled to sequence number 1. In the output block other sequence numbers may be used.
If a name is defined in the input block for sepcomp, this name is also known in the postprocessing program seppost.

END (mandatory), defines the end of the "CONSTANT" block.

The block CONSTANTS must always be read as first block.

4.4 Input for the mesh generator

The input for the mesh generator must be opened with MESH1D, MESH2D or MESH3D, depending on whether the problem is one-, two- or three-dimensional, and must be closed with END.

The records must be given in the order as specified.
An option is indicated like this [option].

MESHnD (mandatory)
opens the input for SEPMESH, and defines the dimension of the space NDIM. (NDIM = n).

COARSE (UNIT=*u*) (optional)
defines that coarseness is used, *u* defines the unit length. Default value: 1.

POINTS (mandatory)
defines the points. Must be followed by records of the type:

```
P1 = ( x_1 , y_1 , z_1, c )
P2 = ( x_2 , y_2 , z_2, c )
           .
           .
           .
Pi = ( x_i , y_i , z_i, c )
```

with i the point number and x_i, y_i and z_i the co-ordinates of point i. For one-dimensional problems only x_i is required, etc. Default values for the co-ordinates: 0.

c must only be used when the COARSE has been read. It defines the coarseness of the elements in the neighbourhood of the point Pi; default value: 1.
Elements sides that contain user point Pi as nodal point, get a local length of approximately cu, where u is the unit defined in the COARSE record.

CURVES (mandatory)
defines the curve. Must be followed by records of the type:

```
C1 = LINE  1 ( P1, P2, NELM=4  )
C2 = ARC   2 ( P1, P2, P3, NELM = 3, RATIO = 1, FACTOR = .3 )
```

etc.
with Ci the curve number.
The names LINE1, LINE2, ARC1, ARC2, USER1 and USER2 are names that may **not** be removed. The following possibilities are available:

```
Ci = LINE  <element_type> ( P1, P2, NELM=n [, RATIO=r, FACTOR=f ] )
Ci = ARC   <element_type> ( P1, P2, P3, NELM=n [, RATIO=r, FACTOR=f ])
Ci = USER  <element_type> ( P1, P2, P3, . . . , Pn )
Ci = CLINE <element_type> ( P1, P2 [,NODD=o] )
Ci = CARC  <element_type> ( P1, P2, P3 [,NODD=o] )
Ci = CURVES ( Ck, Cl, Cm, . . . )
```

with LINE, ARC, USER, CLINE, CARC and CURVES as defined in 4.1.2.
NELM=n gives the number of elements that must be created along the curve (linear or quadratic depending on the value of $<$ *element_type* $>$).
If $<$ *element_type* $>$ is omitted, linear elements are created.

RATIO=r indicates the options for distribution of the nodal points. Possibilities:

r=0: equidistant grid size (default)

r=1: the last cell is f times the first one.

r=2: each next cell is f times the preceding one.

FACTOR=f the factor to be used when r=1 or r=2. Default: f=1.

When LINE is used, a line is generated from point P1 to point P2, for example P3, P6 or P7, P1.

When ARC is used an arc is generated from point P1 to P2 with centre P3. When P3 is given the arc is created counter clockwise, when -P3 is given it is created clockwise.

When USER1 is used, the curve is defined by the points P1, P2, P3, ..., Pn in that sequence, when USER2 is used the curve is defined by the same points, but also the midpoints are generated exactly in the middle of the lines (P1, P2), (P2, P3), . . . , (Pn-1 , Pn).

When CLINE is used, a line is generated from P1 to P2, where the coarseness as given in the points P1 and P2 is used to define the elements. The value of o indicates whether the number of end points of the elements is free, odd or even.
Possibilities:

o=0,1: free

o=2 : number of end points odd

o=3 : number of end points even

Default value: o=0.

When CARC is used, an arc is generated from point P1 to P2 with centre P3, where the coarseness as given in the points P1 and P2 is used to define the elements. For the value of o, see CLINE. When P3 is given the arc is created counter clockwise, when -P3 is given it is created clockwise.

When CURVES is used a curve is defined by the subsequent curves Ck, $Cl, Cm, \ldots$. All these curves must have the same shape number. When the sign of the curve number is positive, the positive direction is used, otherwise (negative sign), the reversed direction of the curve is used. The numbers k, l, and m must be smaller than i.

SURFACES (optional)
defines the surfaces. Must be followed by records of the type:

```
S1 = GENERAL 3 ( C1, C2, C3, C4, . . . )
S2 = GENERAL 5 ( -C5, C6, -C9, C5, . . . )
S3 = QUADRILATERAL 3 ( C4, -C6, C8, C2)
```

etc.
with Si the surface number.
The names GENERAL and QUADRILATERAL are names that may **not** be removed. The value of $<$ *element_type* $>$ gives the shape number of the elements to be created, see Table 4.1.1 ($3 \leq <$ *element_type* $> \leq 6$). Possibilities:

3 Linear triangle with 3 points

4 Isoparametric triangle with 6 points

5 Quadrilateral with 4 points

6 Isoparametric quadrilateral with 9 points

If the user wants to define several element groups, for example since he uses different values of a specific coefficient for different parts of the region, he has to use the options MESHLINE and/or MESHSURF. If the mesh contains elements of different shapes, for example line elements and surface elements, or triangles and quadrilaterals it is also obliged to introduce element groups and hence use the options MESHLINE and/or MESHSURF.

MESHLINE (optional)
defines the one-dimensional elements, or line elements in R^2 and R^3. Must be followed by records of the type:

```
LELM1 = ( SHAPE = 1, C2, C4 )
LELM2 = ( SHAPE = 2 ,C1 )
LELMi = ( SHAPE = j, C6 )
```

with i the element group number. Standard elements must be generated with increasing element group number, first all line elements, then all surface elements, and finally the volume elements.
SHAPE = j defines the shape number of the standard element. See 4.1.4
C1,C2 : line elements are generated along the curves C1 to C2, when C2 is not given only curve C1 is used.

MESHSURF (optional)
defines the two-dimensional elements in R^2 or surface elements in R^3. Must be followed by records of the type:

```
SELM i = ( S1, S2 )
```

with i the element group number. Standard elements must be generated with increasing element group number, first all line elements, then all surface elements, and finally the volume elements. Elements of the same element group must have exactly the same shape, i.e. they must be all linear triangles or all bi-quadratic quadrilaterals and so on.

S1, S2: the elements generated on the surfaces S1, S1 + 1,. . ., S2 are appended to the mesh. When S2 is not given only surface S1 is used.

Special purpose elements

MESHCONNECT (optional)
defines elements that connect user points or nodal points in curves or surfaces.
Also it is possible to connect elements at curves or surfaces. In this way higher dimensional connection elements arise.

These elements are necessary for periodical boundary conditions, see Sections 2.4.3 and 5.4.1. MESHCONNECT must be followed by records of the type:

```
CELMi = POINTS ( P1, P2 )
CELMi = CURVES ( C1, C2 )
```

with i the element group number. Standard elements must be generated with increasing element group number, first all line, surface and volume elements, and then the connected elements.

When CELMi = POINTS (P1 to P2) is defined, an element is created from user point P1 to user point P2.
When CELMi = CURVES (C1 to C2) is defined, elements are defined from nodal points on curve |C1| to nodal points on curve |C2|, or from elements at curve |C1| to elements on curve |C2|. When C1 is positive the elements are created in forward direction starting from the first

position, when C1 is negative the elements are created in reversed order. The same rules are valid for C2.

These connection elements are treated as special elements, which also implies that they are skipped in the postprocessing part.

Auxiliary commands

PLOT (optional)
indicates that the points, curves, the surfaces and the mesh must be plotted, each on a new picture. This record may contain data. In that case it has the following shape:

```
PLOT ( PLOTFM = l, YFACT = y )
```

with PLOTFM = l the length of the plot in centimeters. The default value is machine dependent, usually 20 or 15 centimeters are used.
YFACT = y: Scale factor; all y-coordinates are multiplied by y before plotting the mesh. $y \neq 1$ should be used when the co-ordinates in x and y direction are of different scales, and hence the picture becomes too small. Default value: 1.

END (mandatory)
End of the input for subroutine MESH.

Remark:

The input must be given in the sequence:
MESH card
POINTS
CURVES
SURFACES
VOLUMES
MESHLINE
MESHSURF
MESHCONNECT
PLOT
END

When MESHLINE nor MESHSURF are given, it is supposed that there is only one type of internal element, with element group number 1. This element is a line element when no surfaces are defined. Otherwise it is a surface element. Of course the shapes of elements in different submeshes must be equal.

Example

Consider the region in Figure 4.4.1 Let the number of elements along each side be equal to 10, with equidistant mesh sizes. Then the following input can be used:

```
mesh2d
   points
      p1=(0,0)
      p2=(1,0)
      p3=(1,1)
      p4=(0,1)
   curves
```

```
      c1=line(p1,p2,nelm=10)
      c2=line(p2,p3,nelm=10)
      c3=line(p3,p4,nelm=10)
      c4=line(p4,p1,nelm=10)
   surfaces
      s1 = general3(c1,c2,c3,c4)
   plot
end
```

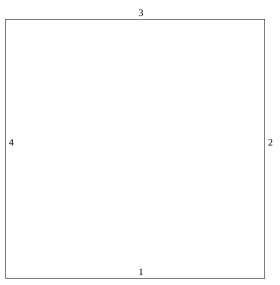


Figure 4.4.1: Example of a region to be divided in elements

An alternative possibility is:

```
mesh2d
   coarse(unit=1)
   points
      p1=(0,0,0.1)
      p2=(1,0,0.1)
      p3=(1,1,0.1)
      p4=(0,1,0.1)
   curves
      c1=cline(p1,p2)
      c2=cline(p2,p3)
      c3=cline(p3,p4)
      c4=cline(p4,p1)
   surfaces
      s1 = general3(c1,c2,c3,c4)
   plot
end
```

Figure 4.4.2 shows the result of the mesh generation. Figure 4.4.3 shows the result of the same region with GENERAL replaced by QUADRILATERAL.

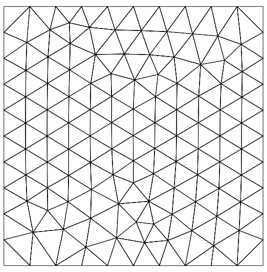


Figure 4.4.2: The result of the mesh generation.

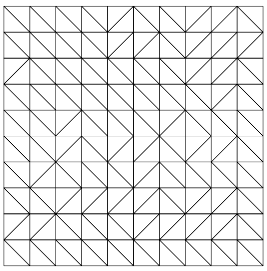


Figure 4.4.3: Result of the same region, with GENERAL replaced by QUADRILATERAL.

4.4.1 Subroutine FUNCCV

Description

Subroutine FUNCCV is used when curves must be generated using the PARAM or CPARAM mechanism. With this subroutine the user may define a curve as function of a parameter t. FUNCCV must be written by the user.

Heading

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

Parameters

DOUBLE PRECISION T, X, Y, Z

INTEGER ICURVE

ICURVE Curve number. Subroutine MESH gives ICURVE the sequence number of the curve to be generated.

T Parameter t for the definition of the curve. Program SEPMESH gives t values between t_0 and t_1.

X,Y,Z the user must give X, Y and Z the values of the co-ordinates as function of the parameter t and the curve number ICURVE.

Input

Program SEPMESH gives ICURVE and T a value

Output

The user must fill the co-ordinates X, Y and Z.

Interface

Subroutine FUNCCV must be programmed as follows:

```
subroutine funccv ( icurve, t, x, y, z)
implicit none
integer icurve
double precision t, x, y, z
        .
        .
        .         statements to give x,y and z a value as function
        .         of t and icurve
        .
end
```

5 The computational part of SEPRAN

5.1 Introduction

The computational part of SEPRAN consists of the program SEPCOMP. In case of standard applications it is sufficient to use SEPCOMP in exactly the same way as SEPMESH. Hence the user creates an input file and he calls program `sepcomp` with this input file as input.
In the case that a functions subroutine has to be added for example to compute space dependent boundary conditions, is is necessary to introduce a local program sepcomp.
This program sepcomp consists of three lines only and is treated in Section 5.2. For the lab at Delft University it is obliged to program your own element and to add an element subroutine. So now it is always necessary to add the main program SEPCOMP.
In Section 5.2 it is described how program SEPCOMP looks like and how you must compile, link and run this program.
Section 5.3 recalls some things you have to look after when programming in FORTRAN.
The input for program SEPCOMP is treated in Section 5.4.
Section 5.5 deals with function subroutines that may be required in case of space or time-dependent quantities.
Finally in Section 5.6 it is described how you must program your own element subroutine.

5.2 How to use program SEPCOMP

With SEPCOMP it is possible to solve relatively complex problems. However, in this introduction we restrict ourselves to the simple case in which the user wants to solve one stationary linear problem or one stationary non-linear problem. In a number of cases it is sufficient to run SEPCOMP as described in Section 3.2. However, if the user wants to supply function subroutines, for example to describe essential boundary conditions or to describe position dependent coefficients, it is not possible to use SEPCOMP immediately. In that case the user must create a simple main program consisting of 3 lines only and also provide the FORTRAN sources for the function subroutines. For the "numerical analysis lab" this is always necessary since a subroutine ELEMSUBR must be provided.

The main program has the following structure:

```
program example
call sepcom ( 0 )
end
```

Subroutine SEPCOM is in fact the body of program SEPCOMP. It has one parameter, which in the standard case must be equal to 0. For the meaning of this parameter the reader is referred to the programmers guide. The name of the program (in this case EXAMPLE) may be chosen freely.

If one or more function subroutines are provided the easiest way is to put these subroutines immediately behind the main program. So in that case we get:

```
program example
call sepcom ( 0 )
end

function func ( ... )
   .
   .
   .
end

function funcbc ( ... )
   .
   .
   .
end

subroutine elemsubr ( ... )
   .
   .
   .
end
```

In this example the parameters and the body of the function subroutines have intentionally been skipped, they are treated in the Sections 5.5 and 5.6.

The main program and the subroutines must be created by a text editor and put into a file. This file must have the extension .f, for example

```
sepcomp.f
```

The user input must be stored in a separate file. In Section 5.3 some general remarks and recommendations about the programming in FORTRAN are given. Inexperienced FORTRAN programmers are advised to read this section and follow the recommendation carefully. They may be of help to avoid errors.

Once the file containing main program and subroutines has been created, this file must be translated (compiled) and the program must be linked with the SEPRAN libraries. Both actions may be performed in one step by the command seplink:

```
seplink file
```

where *file* is the name of the file containing the program without the extension .f.
For example the command:

```
seplink sepcomp
```

compiles and links the file `sepcomp.f`.

In the first step of seplink the fortran code is checked and translated. Fortran error-messages appear on the screen. If there are too much errors, it may be necessary to write these messages to a file for later inspection.
This may be done by redirecting to an output file like:

```
seplink sepcomp >& outputfile
```

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 funcbc instead of funcbc you get the message

```
undefined symbol

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. If both compilation and linking have been carried out successfully seplink produces a file with the name of the seplink parameter (that is without the extension .f). So `seplink sepcomp` produces a file `sepcomp`. To run the program `sepcomp` in a unix environment you type:

```
sepcomp <  inputfile  >  outputfile
```

or

```
sepcomp <  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.

The main program uses the file `meshoutput` generated by sepmesh and produces two files `sepcomp.inf` and `sepcomp.out` that will be used by seppost.

Remark: the outputfile may have any name except `meshoutput`, `sepcomp.inf`, `sepcomp.out` or sepplot.$* * *$.

To avoid unnecessary typing a body of program sepcomp with corresponding subroutine elemsubr is available.
You can get this file (`sepcomp.f`) locally by giving the command:

```
   sepgetpract sepcomp
```

If you use another name then sepcomp, the body is put into a file with that new name with suffix `.f`. For example `practicum` creates a file `practicum.f`.
The file sepcomp.f can be edited using nedit, to get the correct program.

Remark:

The use of sepgetpract is very important: not only does it prevent unnecessary typing, also the number of errors in your program may be largely reduced.

5.3 Programming considerations

SEPRAN consists of a set of FORTRAN subroutines that can be used in standard FORTRAN 77 programs. If you write your own subroutines or function subroutines, it is advised to follow the next recommendations:

(i) In the main program and each subroutine it is advised to declare all variables explicitly. To check the declarations put the next statement immediately after the program or subroutine statement:

```
implicit none
```

Furthermore, all reals must be declared double precision, because SEPRAN computes only in double precision in order to avoid loss of accuracy.

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.

(ii) The following FORTRAN conventions are standard:

A C 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.

(iii) The input from the standard input file is organized in records. A record is a line. Records must always be at most 80 positions long.
SEPRAN requires a special form of input.

For a description of the rules that apply see Section 4.3

Mark that you need two files for the computational program, the fortran file satisfying all standard Fortran rules and the input file satisfying the SEPRAN rules.
The fortran file has the extension `.f` (usually it is called `sepcomp.f`).
The name of the input file is free, but a common extension is `.prb`.

List of frequently made errors

- The FORTRAN text starts before column 7 or ends behind column 72.
- The input file is put in the same file as the fortran file
- The quotient of two integers is computed, like `1/4`. According to FORTRAN rules, the result is an integer that is chopped, hence `1/4 = 0`.

5.4 Description of the input for program SEPCOMP

Before describing the input block, we consider a simple example.

A sample input file

Consider the square region as defined in Section 4.2. Suppose that we want to solve the Poisson equation:

$$-\Delta c = f \tag{5.4.1}$$

with f a given function. In our example we choose $f = 1$. Let the boundary conditions be $c = 1$ on curve C_1 and $\frac{\partial c}{\partial n} = 0$ on the rest of the boundary.
This problem is part of the general class of second order elliptic equations as described in the manual Standard problems, Section 3.1. According to this manual the boundary condition $\frac{\partial c}{\partial n} = 0$ is natural and does not need to be prescribed. It is satisfied automatically.
The Poisson equation corresponds to the general case with $a_{11} = 1$ and $a_{22} = 1$. It concerns the sixth and ninth coefficient respectively. The source term is given by coefficient 16.
The matrix is symmetrical and positive definite.
The following input file may be used as input for program SEPCOMP:

```
#
#  square.prb
#
#  Example file for the SEPRAN introduction, Section 5.4
#
#  Define some constants for the problem, See introduction, Section 4.3
#
constants
   reals
      a11 = 1                        # coefficient for the differential equation
      a22 = 1                        # coefficient for the differential equation
      f   = 1                        # source term
end
#
#   Problem definition, it is described what type of problem it concerns
#   See introduction, Section 5.4.1
#
problem
   types                             # Define the type of equations
      elgrp1 = (type=800)            # General second order elliptic equation
   essboundcond                      # Define where essential boundary
                                     # conditions are given (not the value)
      curves(c1)                     # essential boundary conditions on c1
end
#
#   Define the structure of the large matrix
#   See introduction, Section 5.4.2
#
matrix
   method = 1                        # The matrix is symmetrical
                                     # It is stored as a profile matrix hence
                                     # a direct solver is used
end
#
#   Define the values of the non-zero essential boundary conditions
#   See introduction, Section 5.4.3
```

```
#
essential boundary conditions
   curves(c1), value=1              # u=1, along curve c1
end
#
#   Define the coefficients for the differential equation
#   See introduction, Section 5.4.4 and
#   manual Standard Problems Sections 3.1
#
coefficients
   elgrp1 (nparm=20)            # The element group has at most 20 coefficients
      coef 6 = $a11             # coefficient a11 (Laplace)
      coef 9 = $a22             # coefficient a22 (Laplace)
      coef16 = $f               # source term
end
#
#   Information for the linear solver
#   See introduction, Section 5.4.6
#
solve
   positive definite             # The matrix is positive definite
end
end_of_sepran_input
```

General rules

The input for program SEPCOMP is subdivided into a number of blocks. Some of these blocks must be given in a fixed sequence; all others are free. Each block starts with a specific main keyword and ends with the keyword END. Unless stated otherwise all commands in a block must be given on a new line. It is advised to indent the input between main keyword and the keyword END to make the block more visible. The same is advised for subblocks. The end of the input is indicated by the physical end of file or by the keyword END_OF_SEPRAN_INPUT. This last keyword may be necessary if the user reads his own input in the standard SEPRAN input file.
SEPCOMP starts with reading all SEPRAN input before carrying out the necessary computations. In this way input errors are checked immediately. The present version of SEPCOMP recognizes the following blocks at least the following main keywords indicating the beginning of a block:

- PROBLEM
- MATRIX
- ESSENTIAL BOUNDARY CONDITIONS (3 keywords)
- CREATE
- SOLVE
- NONLINEAR EQUATIONS (2 keywords)
- DERIVATIVES
- INTEGRALS
- BOUNDARY_INTEGRAL
- OUTPUT
- STRUCTURE

Other keywords that may be interpreted are described in the SEPRAN users manual. The block PROBLEM must be given as first block, it may only be preceded by the block START, which is not treated in this manual. All other blocks may be given in any sequence. The information of a block, however, must always be positioned between the main keyword and the keyword END. The block PROBLEM is mandatory, all other blocks are optional.
If no input for a block is given default values are used.
The main blocks have the following meaning:

PROBLEM Defines the type of problem to be solved, i.e. the type of differential equation, the type of boundary conditions, at which boundaries these boundary conditions are given etc. PROBLEM only defines types not values. So in the part PROBLEM it is fixed at which boundaries essential boundary conditions must be prescribed, but not what the values of these boundary conditions are. See Section 5.4.1.

MATRIX Defines the type of storage to be used for the large matrix. In this part it is given whether the large matrix is symmetrical, complex, etc. But also the user defines whether the storage scheme corresponds to a direct method or a compact method. Implicitly this defines the type of solver that will be used to solve the systems of linear equations. If a direct storage is used, a profile solver will be called (direct method), if a compact storage is used, the linear system is solved by an iterative method.
If the part MATRIX is skipped it is assumed that the matrix is real, non-symmetric and that a direct method is used. See Section 5.4.2.

ESSENTIAL BOUNDARY CONDITIONS Defines the values of the essential boundary conditions. It is only necessary to define the non-zero essential boundary conditions, all other essential boundary conditions are made equal to zero automatically.
If the part ESSENTIAL BOUNDARY CONDITIONS is skipped all essential boundary conditions are set equal to zero. See Section 5.4.3.

CREATE Can be used to define a vector, for example the start vector in case of a non-linear problem. See Section 5.4.4.

SOLVE Gives information with respect to the linear solver to be used. For example in the case of a direct method, it is possible to tell the solver that the matrix is positive definite. In the case of an iterative solver, the user may give extra information about the type of linear solver etc.
If the part SOLVE is skipped, the default values are used. This means that in the case of a storage scheme corresponding to a direct solver, a profile method is used and it is not assumed that the matrix is positive definite. In the case of an iterative solver this means that the default iterative solver, with the default accuracy and the default set ups is used. See Section 5.4.5.

NONLINEAR EQUATIONS Indicates that the partial differential equation to be solved is stationary and non-linear. In that case an iterative procedure is necessary to solve the non-linear problem. In each step of the non-linear iteration a linear system of equations is solved. In this part the user gives some information about the iteration process.
If the keywords NONLINEAR EQUATIONS are skipped it is assumed that the partial differential equation to be solved is linear and no iteration is carried out. See Section 5.4.6.

DERIVATIVES This keyword is used when a derived quantity of the solution must be computed, for example the gradient of the solution.
This keyword is only activated in combination with a structure block. See Section 5.4.8.

INTEGRALS This keyword is used when an integral over the solution must be computed.
This keyword is only activated in combination with a structure block. See Section 5.4.9.

BOUNDARY_INTEGRAL This keyword is used when an integral over (a part) the boundary of the the solution must be computed.
This keyword is only activated in combination with a structure block. See Section 5.4.10.

OUTPUT Defines which output is written to the file sepcomp.out.
This output may be used in the post-processing part of SEPRAN.
If the keyword OUTPUT is skipped only the computed solution is written to the output file. Otherwise it is also possible to compute derivatives or other derived quantities and to write these to the file sepcomp.out See Section 5.4.7.

STRUCTURE This keyword is very special. In fact it defines which actions must be carried out in the program and in which sequence. If a standard linear or non-linear problem must be solved, there is no need to give the structure block. However, as soon as something extra is required, like an integral of derivates that must be computed, or if some prints during the computations must be made, it is necessary to supply this extra block. See Section 5.4.11.

So a typical input for a linear stationary problem may look like:

```
problem
  .
  .
  .
end
matrix
  .
  .
  .
end
essential boundary conditions
  .
  .
  .
end
solve
  .
  .
  .
end
```

and a typical input for a nonlinear stationary problem:

```
problem
  .
  .
  .
end
matrix
  .
  .
  .
end
essential boundary conditions
  .
  .
  .
end
solve
  .
  .
  .
end
nonlinear equations
  .
  .
  .
end
```

In the next subsections the input of each of the blocks is described.

5.4.1 The main keyword PROBLEM

The block defined by the main keyword PROBLEM defines which problem is to be solved by program SEPCOMP. For each element group defined in SEPMESH the user must indicate what type of problem has to be solved. Problems are indicated by so-called type numbers.
SEPRAN also allows for the definition of your own elements. For that reason the group of element numbers between 1 and 99 is strictly reserved for user defined elements, whereas type numbers larger than 99 correspond to SEPRAN standard elements. Type numbers smaller than 1 have a special meaning.
Type number -1 is used for periodical boundary conditions. See also Section 2.4.3. This type number may only be used for element s defined by MESHCONNECT as described in Section 4.4.

For the lab you have to use type numbers between 1 and 99 and eventually -1.
If type numbers between 1 and 99 are used the user must provide his own element subroutines as described in Section 5.5. For each differential equation it is necessary to give boundary conditions. SEPRAN distinguishes between so-called essential boundary conditions and natural boundary conditions. An essential boundary condition is a boundary condition that prescribes unknowns at the boundary explicitly, natural boundary conditions in general give some information about derivatives or combinations of unknowns and derivatives at the boundary. Before using SEPRAN, the student himself must decide which boundary conditions are natural.

Natural boundary conditions require extra elements, the so-called boundary elements. These elements may be defined in the part PROBLEM as boundary elements.

The block defined by the main keyword PROBLEM has the following structure:

```
PROBLEM
   TYPES
     data corresponding to TYPES
   NATBOUNCOND
     data corresponding to NATBOUNCOND
   BOUNELEMENTS
     data corresponding to BOUNELEMENTS
   ESSBOUNDCOND
     data corresponding to ESSBOUNDCOND
END
```

The keywords PROBLEM, END and TYPES are mandatory. All subkeywords may be given in arbitrary order as long as they appear only once. The data corresponding to these subkeywords must be given immediately after the keywords themselves.
If the keyword NATBOUNCOND is given then also the keyword BOUNELEMENTS must be present.

Explanation of the subkeywords and description of the records (options are indicated between the square brackets "[" and "]"):

PROBLEM (mandatory)
opens the input for this block.

TYPES (mandatory)
defines the problem definition numbers of the standard elements. Must be followed by records of the type:

```
ELGRP 1 = (type = n1)
ELGRP 2 = (type = n2)
ELGRP i = (type = n3)
```

with i the element group number; exactly number of element groups (NELGRP) data records are necessary. ni is the problem definition number of the i^{th} element group.
The element group number refers to the element group number defined in the mesh generation part. The number of element groups to be defined in this part TYPES must be exactly equal to the number of element groups defined in the mesh generation.
The type number is used to define which type of problem must be solved. This type number is available in the element subroutine, where it can be used to distinguish between different element types. For the lab only type numbers between 1 and 99 may be used.
Type numbers less than 1 have a special meaning.
If the number of degrees of freedom per point is not equal to 1 then each record with `ELGRP` must be followed by a record with

```
NUMDEGFD = n
```

where n is the number of degrees of freedom per point in that element. For almost all exercises there is no need to give this statement.

NATBOUNCOND (optional)
indicates that standard boundary elements are used. Must be followed by data records of the type:

```
BNGRP 1 = (type = n1)
BNGRP i = (type = ni)
```

with i the boundary element group number and ni is the boundary problem number of the i^{th} boundary element group.
The boundary element groups must be defined sequentially from 1. No boundary element group numbers may be skipped. The largest boundary element group number defines the number of boundary element groups (NUMNATBND).
Internally in the element subroutines the boundary groups get as element sequence number NELGRP + IBNGRP, where IBNGRP is the boundary element group sequence number and NELGRP is the number of element groups defined in the mesh generation.

BOUNELEMENTS (must only be used when NATBOUNCOND is used)
indicates that boundary elements are created. Must be followed by records of the following type:

```
BELM1 = POINTS ( P3, P6, P8, . . . )
BELM2 = CURVES ( C1 to C2 )
BELMi = CURVES ( C5 )
```

These records take care of the generation of boundary elements.
i is the boundary element group number; i may be used more than once. If boundary element group numbers are not used, the number of elements for that group is equal to zero. The boundary elements must be created with increasing boundary element group number. When the boundary elements consist of points, the function POINTS must be used followed by the numbers of the user defined points (see subroutine MESH). Only points that coincide with nodal points may be used. At most 20 points are permitted in one record.
When the boundary elements consist of curve elements, the function CURVES must be used, followed by the curve numbers.
C1 to C2: means that boundary elements are generated along the curves C1 to C2, when C2 is not given only curve C1 is used. When C2 is given, the curves C1 to C2 must be subsequent curves with coinciding initial and end point, i.e. the end point of C1 must be equal to the initial point of C1 + 1 etc.
The boundary elements must always be created counter-clockwise with respect to the inner region. Hence the corresponding curves must also be generated counter-clockwise.

The boundary elements must be created in the sequence: points, curves, surfaces. For surface boundary elements (R^3 only) the user is referred to the programmers guide.
For each boundary element group defined before it is necessary to create boundary elements.

ESSBOUNCOND (optional)
indicates that essential boundary conditions will be prescribed. In this part it is described in which positions we have essential boundary conditions and which unknowns are prescribed. However, the values of these boundary conditions are not yet given. They are described by either the separate command ESSENTIAL BOUNDARY CONDITIONS or by CREATE VECTOR. Both do not belong to the part PROBLEM. If, however, a degree of freedom is not identified as essential boundary condition in this part of the input, it will never become an essential boundary condition and values defined in other parts of the input given to these unknowns will never be recognized as essential boundary conditions.
This record must be followed by records of the type:

```
DEGFD1, DEGFD3 = POINTS ( P1, P5, P8 )
DEGFD2 = POINTS ( P2, P3 )
DEGFD1, DEGFD2, DEGFD3 = CURVES ( C1 to C5 )
```

These records must be given in the sequence POINTS, CURVES, SURFACES.
DEGFDj indicates that the j^{th} degree of freedom will be prescribed (the value of these degrees of freedom are filled by the block ESSENTIAL BOUNDARY CONDITIONS).

Hence DEGFD1, DEGFD3 indicates that the first and third degree of freedom in the corresponding nodal points are prescribed. At most 20 degrees of freedom are permitted in one record. When DEGFDj = is omitted all degrees of freedom are supposed to be prescribed in the corresponding nodal points.

When the essential boundary conditions are given in user defined points, the function POINTS must be used followed by the numbers of the user defined points (see 3.1.2). Only points that coincide with nodal points may be used. At most 20 points are permitted in one record.
When essential boundary conditions are given on curves the function CURVES must be used followed by the curve numbers C1 to C5 indicating that essential boundary conditions of this type are defined on the curves C1 to C5, or C1 only when C5 is omitted. When C5 is given, the curves C1 to C5 must be subsequent curves with coinciding initial and end point, i.e. the end point of C1 must be equal to the initial point of C1 + 1 etc.

END (mandatory)

5.4.2 The main keyword MATRIX

The block defined by the main keyword MATRIX defines the structure of the large matrix and hence implicitly the linear solver to be used. SEPRAN distinguishes between symmetric and non-symmetric, real and complex matrices. Furthermore storage schemes for direct methods differ from the storage scheme for iterative solvers.
Whether the large matrix is symmetrical or not depends on the type of problem to be solved. In the manual STANDARD PROBLEMS for each problem it is given whether the matrix is symmetrical or not. This is also the case for real and complex matrices. Each symmetrical matrix may of course be stored as a non-symmetrical matrix, however, the storage needed doubles in general and also the computation time may increase. A real matrix may in general not be stored as a complex matrix.
The choice between a direct linear solver and an iterative linear solver is not so easy to make. In general a direct solver is the most robust and most simple to use. However, for large problems in R^2 and smaller problems in R^3 iterative solvers use much less memory and often also less computation time. However, for some problems iterative solvers converge very slowly or even diverge. Unfortunately no hard criterion can be formulated when one method is preferred above the other one. An important remark is that in the case of time-dependent problems and sometimes also stationary non-linear problems in general a good initial estimate of the solution is available. In combination with a not too strict termination criterion this makes the iterative solvers more favourable.

The block defined by the main keyword MATRIX has the following structure (options are indicated between the square brackets "[" and "]"):

MATRIX (mandatory)
indicates that information of the structure of the large matrix will be given.

METHOD = i (mandatory), gives information of the structure of the large matrix. Depending on the value of i the system of equations is solved by a direct solution method (Gaussian elimination) or by an iterative method.
Possible values for i are for example:

1-4 The matrix is stored as a so-called profile matrix, which implies that a direct solution method is used.

- 1 The matrix is real symmetric.
- 2 The matrix is real (in general not symmetric).
- 3 The matrix is complex symmetric.
- 4 The matrix is complex (in general not symmetric).

5-8 The matrix is stored as a so-called compact matrix, which means that an iterative solution method is used.

- 5 The matrix is real symmetric.
- 6 The matrix is real (in general not symmetric).
- 7 The matrix is complex symmetric.
- 8 The matrix is complex (in general not symmetric).

Other values of i are not treated in this manual.

END (mandatory)
end of the input of the block MATRIX.

Remark: if the block corresponding to MATRIX is skipped METHOD = 2 is assumed.

5.4.3 The main keywords ESSENTIAL BOUNDARY CONDITIONS

The block defined by the main keywords ESSENTIAL BOUNDARY CONDITIONS defines whether the solution vector is real or complex and also defines the values of the essential boundary conditions. At which boundaries essential boundary conditions are given has already been described in the part PROBLEM. In fact all essential boundary conditions that are not explicitly given in this part are set equal to zero.

The block defined by the main keywords ESSENTIAL BOUNDARY CONDITIONS has the following structure (options are indicated between the square brackets "[" and "]"):

ESSENTIAL [COMPLEX] BOUNDARY CONDITIONS (mandatory)
opens the input for PRESDF.

The option COMPLEX indicates that the solution vector is a complex vector.

Must be followed by records defining the essential boundary conditions. Only the non-zero essential boundary conditions must be specified in this part. If all essential boundary conditions are zero, no extra records are necessary.

Essential boundary conditions in user points and curves may be defined as follows:

```
POINTS ( P1, P5, P8 ), DEGFD1 = (VALUE = 1.5)
POINTS ( P1 ), DEGFD2 = (FUNC = 3)
CURVES ( C1 to C5 ), DEGFD1 = (VALUE = 3D0)
```

POINTS (P*i*, P*j*, P*k*) means that essential boundary conditions are prescribed in the user points P*i*, P*j* and P*k*. The brackets (and) surrounding P*i*, P*j* and P*k* may **not** be omitted, even if only one user point is given.

CURVES *i* **(C1 to C5)** indicates that essential boundary conditions are prescribed on the curves C1 to C5, or only C1 if C5 is omitted. If C5 is given, the curves C1 to C5 must be subsequent curves with coinciding initial and end point, i.e. the end point of C1 must be equal to the initial point of C1+1 etc.
The brackets surrounding C1, C5 may **not** be removed.

DEGFD*i* = means that the i^{th} degree of freedom is prescribed by this record. If omitted, the first degree of freedom is prescribed.

VALUE = 1.5 indicates that the degrees of freedom defined in this data record get the value 1.5.

FUNC = 3 indicates that the degrees of freedom defined in this record are given by a function depending on the co-ordinates. In that case the user must submit a function subroutine FUNCBC as described in 5.5.1. The value following the equals sign corresponds to the parameter IFUNC in FUNCBC, hence in this example IFUNC in the call of FUNCBC is equal to 3.

The brackets surrounding VALUE = .. or FUNC = .. are essential.
For other values of interest the reader is referred to the Programmers Guide.

END (mandatory)
end of the block ESSENTIAL BOUNDARY CONDITIONS.

Remark: For complex vectors a complex value may be denoted by VALUE=(a,b), with a and b two reals. The brackets are mandatory in this case! VALUE=a defines a real boundary condition.

For complex vectors FUNC=k refers to a subroutine CFUNCB instead of FUNCBC. See 5.5.2 for a definition of CFUNCB.

Remark: if the block corresponding to ESSENTIAL BOUNDARY CONDITIONS is skipped it is assumed that all essential boundary conditions have the value 0 and moreover that the solution vector is real. So in case of a complex problem always the part ESSENTIAL BOUNDARY CONDITIONS must be given.

5.4.4 The main keyword CREATE

The block defined by the main keyword CREATE is used to create a SEPRAN vector, which may be a solution vector or a vector of special structure. If this block is available it is always read and interpreted. However, the actual creation of the vector takes only place if the option CREATE_VECTOR is used in the input block "STRUCTURE".

The block defined by the main keyword CREATE has the following structure (options are indicated between the square brackets "[" and "]"):

```
CREATE [COMPLEX] VECTOR [,SEQUENCE_NUMBER = s]
   Records defining the output vector
END
```

The various options in the CREATE record have the following meaning:

CREATE VECTOR mandatory, means that a vector must be created.

COMPLEX indicates that the solution vector is a complex vector.

SEQUENCE_NUMBER $= s$ may be used to distinguish between various input blocks with respect to the creation of vectors.

After the CREATE keyword, records defining the vector must be given. The vector is created by applying the definitions sequentially, so for example first a vector may be set to a constant, then the curves may be changed into other values and finally the user points may be changed. The sequence of the commands defines the sequence in which the vector is filled. This sequence may be essential for the final value in a specific node. The records defining the computation have the following shape:

[functional description] [degrees of freedom] [location part] in arbitrary order.

The functional description may be of one of the following shapes:

```
VALUE = alpha
VALUE = (alpha , beta)
FUNC[TION] = k
```

with

VALUE $= \alpha$ sets the required degrees of freedom equal to the constant value α. If the vector is complex, also a complex value may be given like (α, β). In that case $(\alpha, 0)$ and α are identical.

FUNCTION $= k$ defines the degrees of freedom as a function of the co-ordinates. In the case of a real vector the function is defined by the function subroutine FUNC:

```
function FUNC ( k, X, Y, Z )
```

see Section 5.5.3.
In the case of a complex vector the function is defined by the function subroutine CFUNC:

```
function CFUNC ( k, X, Y, Z )
```

see Section 5.5.3. If the functional description is omitted, the default: VALUE=0 is assumed.

The degrees of freedom part may have one of the following structures:

```
DEGFD2
DEGFD3, DEGFD1, DEGFD6
```

which indicates that in the nodes to be created only physical unknown 2 or the physical unknowns 1, 2 and 6 are filled.
If this part is omitted all degrees of freedom in the nodes to be created are filled.

The location part may have one of the following structures:

```
POINTS ( Pk, Pl, . . . , Pm)
USER POINTS ( Pk to Pl )
CURVES (Cj [to Cm] )
```

These records have the following meaning:

POINTS $(P_{i1}, P_{i2}, ...)$ defines all user points between the brackets.

CURVES $[l]$ (C_j [to C_m]) defines only the part of the vector in the curves C_j to C_m (or C_j if C_m is omitted).
The curves C_j to C_m must be subsequent curves!

USER POINTS (P_{l_1} [to P_{l_2}]) defines the part of the vector in the user points P_{l_1} to P_{l_2} (or P_{l_1} if P_{l_2} is omitted).

If this part is omitted, all nodes are used.

Typical examples are:

```
VALUE=3
FUNC=6
VALUE=(2,0.5)

DEGFD1, VALUE=5
FUNC=6, DEGFD1, DEGFD3

POINTS (P1, P2, P6), DEGFD2, FUNC=5
POINTS (P1), FUNC=3, DEGFD1
USER POINTS (P3 to P6)
DEGFD3, USER POINTS (P3 to P7), VALUE=0.5
FUNC=2, DEGFD2, DEGFD6, CURVES 3 (C1 to C3)
```

Remarks:

- If no data records are given after the CREATE command, the complete vector is set equal to zero.
  However, as soon as at least one data record defining the vector or a part of it is given, the vector is not initialized. That means that degrees of freedom that are not defined in the data records are not changed or initialized. The user is responsible for the correct filling of the vector.

- The vector is filled in the order given in the input file. Hence, the statements

  VALUE=0
  DEGFD2 = (FUNC=3)

  set first the vector equal to zero and then replace the second component by the function

defined by FUNC=3.
On the other hand the statements

DEGFD2=(FUNC=3)
VALUE=0

have as final effect that the vector is set equal to zero. In this case the first command is useless and only consumes computing time.

A typical input block "CREATE" might be:

```
CREATE sequence_number = 2
  value = 0
  degfd1 = func = 3
  curves (c1 to c3), degfd2 = func = 4
  curves (c2 to c3), degfd3 = func = 7
END
```

5.4.5 The main keyword SOLVE

The block defined by the main keyword SOLVE gives information with respect to the linear solver to be used. Even in a non-linear problem a series of linear problems is solved, and hence this block makes also sense in that case.
The type of linear solver to be used (direct or iterative) has already been defined in the block MATRIX. In this part some extra information for the solver may be defined.

The block defined by the main keyword SOLVE has the following structure (options are indicated between the square brackets "[" and "]"):

```
solve
   positive_definite
   iteration_method = iter_method [,options]
end
```

The sequence of the subkeywords is arbitrary.

POSITIVE_DEFINITE indicates that the matrix to be solved is not only symmetrical, but also positive definite. This is only used in case of a direct solver. This command may not only improve the computation time, it also offers an extra check on the correctness of the input. This keyword must only be used if a direct solver is applied, i.e. METHOD in the input block MATRIX is between 1 and 4.

ITERATION_METHOD defines the type of iteration method to be used as well as the options to be applied. If the structure of the matrix as defined in the block MATRIX ... END by METHOD =, has got a value between 1 and 4, the input about the iteration method is neglected, since then always a direct solver will be used.
iter_method may take one of the following values

```
cg
cgs
gmres
```

In symmetrical problems (METHOD=5) always the conjugate gradient method is used. If the matrix is non-symmetrical (METHOD=6) the method is defined by *iter_method*.
If CG is given the bi-cgstab method of Sonneveld and van der Vorst is used. CGS activates the conjugate gradients squared method of Sonneveld and GMRES the so-called GMRES method.

The options following *iter_method* may be given in any sequence. They must be given in the same record. If it is not possible to give all options within 80 columns it is necessary to proceed on the next line. In that case the first line must be closed with the continuation mark // i.e slash immediately followed by another slash. This process may be used recursively.

The following options are available:

```
preconditioning = prec
max_iter = m
accuracy = eps
print_level = p
```

PRECONDITIONING defines the type of preconditioner to be used. The following values for *prec* are available:

none no preconditioner is used

diagonal diagonal scaling of the matrix is used as preconditioner

ilu the preconditioner is a so-called incomplete LU decomposition

eisenstat incomplete LU decomposition where only the diagonal is changed efficient implementation of Eisenstat

Gauss_Seidel preconditioning with a Gauss Seidel iteration

mod_eisenstat modified incomplete LU decomposition according to Axelson efficient implementation of Eisenstat

The default value is eisenstat.

MAX_ITER restricts the maximum number of iterations to m. If the number of iterations exceeds this maximum an error message is given and the program is halted.
The default value for m is the number of unknowns, but usually the process should be finished much earlier.

ACCURACY defines when the iteration process is terminated. If the absolute error is less than ϵ the iteration is stopped. The default value is $\epsilon = 10^{-3}$

5.4.6 The main keyword NONLINEAR_EQUATIONS

The block defined by the main keyword NONLINEAR_EQUATIONS indicates that a non-linear stationary problem has to be solved. In this block information concerning the iteration process must be defined.

The block defined by the main keywords NONLINEAR_EQUATIONS has the following structure (options are indicated between the square brackets "[" and "]"):

```
NONLINEAR_EQUATIONS  (optional):  opens the input for the non-linear solver.
   GLOBAL_OPTIONS, options (optional)
END  (mandatory)
```

The sequence of the subkeywords, subsubkeywords and subsubsubkeywords is arbitrary. However, subsubkeywords corresponding to a subkeyword must all be grouped under the subkeyword and so on. All sub, subsub and subsubsub keywords given above must start at a new line in the input file.

The subkeyword GLOBAL_OPTIONS define some global choices with respect to the linear solver. The options itself should be put on the same line as the keyword GLOBAL_OPTIONS. If this line exceeds position 80, continuation at the next line is necessary. This is activated by closing the line by // (before column 81) and putting the rest of the information on the next line. This process may be indefinitely repeated. The following options are available:

```
maxiter = m                  (Default 20)
miniter = m                  (Default 2)
accuracy = eps               (Default 1d-3)
print_level = p              (Default 0)
iteration_method = m         (Default standard)
at_error = e                 (Default stop)
```

Meaning of the various options:

maxiter = m defines the maximum number of iterations that may be performed. If the number of iterations reaches this maximum value and the accuracy has not been reached, an error message is given and the program is terminated.

miniter = m defines the minimum number of iterations that have to be carried out.

accuracy = ϵ defines the accuracy at which the iteration terminates, provided the minimum number of iterations has been performed. Accuracy has been reached if the difference between two succeeding iterations is less than ϵ.

print_level = p gives the user the opportunity to indicate the amount of output information he wants from the iteration process. p may take the values 0, 1 or 2. The amount of output increases for increasing value of p.

iteration_method = m defines the type of non-linear iteration method that is applied. Possible values for m are:

```
standard
newton
```

standard means that a standard iteration method is applied: The process starts with a given start vector $\mathbf{u}^0$ containing the boundary conditions. In each iteration $\mathbf{S}^k\mathbf{u}^{k+1} = \mathbf{f}^k$ is solved, where the solution vector $\mathbf{u}^{k+1}$ also contains the given boundary conditions. The matrix $\mathbf{S}^k$ and the right-hand-side vector $\mathbf{f}^k$ may vary in each iteration step.

newton corresponds to the standard Newton (Raphson) method. This process is as follows:

```
start:   given start vector $\mathbf{u}^0$
While not converged
   Solve correction $\mathbf{S}^k\ \delta\mathbf{u} = \mathbf{f}^k$
   Correct          $\mathbf{u}^{k+1} = \mathbf{u}^k + \delta\mathbf{u}$
```

The correction in each step must satisfy homogeneous essential boundary conditions, since otherwise the essential boundary conditions are changed in the correction step.

at_error = ***e*** defines which action should be taken if the iteration process terminates because no convergence could be found. Possible values are:

```
stop
return
```

If `stop` is used the iteration process is stopped if no convergence is found, otherwise (`return`) means that control is given back to the main program and the result of the last iteration is used as solution.
This option is very suitable to check what happens during the iteration. Suppose that the iteration does not converge and you do not have any idea what causes it.
A possible way to check what happens is to start with `maxiter = 1` in combination with `at_error = return`.
After that you can check the solution at the first iteration using seppost. In this way some possible errors, like incorrect boundary conditions may be detected.
Once this step is correct you may for example proceed with with `maxiter = 2` in combination with `at_error = return`, and check the solution again with seppost. This gives you the opportunity to check your iteration process.

5.4.7 The main keyword OUTPUT

The block defined by the main keyword OUTPUT defines which output must be written to the file sepcomp.out for post-processing purposes. If omitted only the solution is written, otherwise the user may define which derived quantities must be computed and written to sepcomp.out.

The block defined by the main keyword OUTPUT has the following structure (options are indicated between the square brackets "[" and "]"):

```
    OUTPUT (optional):  opens the input for the output part.
```

If more than one vector is created in the computational part, for example if a vector is created separately or if derivatives are computed and stored in a vector a data record of the shape

```
   write n solutions
```

may be used which indicates that n vectors are written to the file `sepcomp.out` to be used by SEPPOST. This can only be used in combination with the STRUCTURE block.
The block must be closed with the keyword:

```
END (mandatory): end of the input of this block
```

If only one vector is to be written, this block may be skipped.

5.4.8 The main keyword DERIVATIVES

The block defined by the main keyword DERIVATIVES gives information with respect to the derived quantities (usually derivatives) to be computed. If this block is available it is always read and interpreted. However, the actual computation of derivatives takes only place if the option DERIVATIVES is used in the input block "STRUCTURE".

The block defined by the main keyword DERIVATIVES has the following structure (options are indicated between the square brackets "[" and "]"):

```
derivatives [,sequence_number = s]
   icheld = k
   seq_input_vector = %name
end
```

The keywords DERIVATIVES and END are mandatory even when there is no subkeyword.

Meaning of the keywords:

DERIVATIVES ,SEQUENCE_NUMBER = *s* opens the input for the computation of derived quantities.
The sequence number *s* may be used to distinguish between various input blocks with respect to the derivatives.

ICHELD = *k* defines the type of derived quantity to be computed. This parameter is passed undisturbed to the element subroutine ELDERVSUBR see Section 5.6.2. This parameter may be used to distinguish between several possibilities.
The default value for ICHELD is 1.

SEQ_INPUT_VECTOR = *%name* defines from which input vector the derivatives must be computed.
The parameter *%name* refers to the vector with name `name`.

END defines the end of the input block

The sequence of the subkeywords is arbitrary.

The output vector is supposed to have the same number of degrees of freedom per point as the solution vector (usually 1).

5.4.9 The main keyword INTEGRALS

The block defined by the main keyword INTEGRALS gives information with respect to the integrals to be computed. If this block is available it is always read and interpreted. However, the actual computation of integrals takes only place if the option INTEGRALS is used in the input block "STRUCTURE".

The block defined by the main keyword INTEGRALS has the following structure (options are indicated between the square brackets "[" and "]"):

```
integrals [,sequence_number = s]
   icheli = i
end
```

The keywords INTEGRALS and END are mandatory even when there is no subkeyword.

Meaning of the keywords:

INTEGRALS ,SEQUENCE_NUMBER = s opens the input for the computation of integrals. The sequence number s may be used to distinguish between various input blocks with respect to the integrals.

ICHELI = k defines the type of integral to be computed. This parameter is passed undisturbed to the element subroutine ELINTSUBR see Section 5.6.3. This parameter may be used to distinguish between several possibilities.
The default value for ICHELI is 1.

END defines the end of the input block

5.4.10 The main keyword BOUNDARY_INTEGRAL

The block defined by the main keyword BOUNDARY_INTEGRAL gives information with respect to the boundary integrals to be computed. If this block is available it is always read and interpreted. However, the actual computation of integrals takes only place if the option BOUNDARY_INTEGRAL is used in the input block "STRUCTURE".

The block defined by the main keyword BOUNDARY_INTEGRAL has the following structure (options are indicated between the square brackets "[" and "]"):

```
boundary_integral [,sequence_number = s]
   ichint = i
   ichfun = j
   irule = k
   curves (c1, c2, c3, ... )
   degree_of_freedom = d
end
```

The keywords BOUNDARY_INTEGRAL and END are mandatory even when there is no subkeyword.

Meaning of the keywords:

BOUNDARY_INTEGRAL , SEQUENCE_NUMBER = s opens the input for the computation of boundary integrals.
The sequence number s may be used to distinguish between various input blocks with respect to the boundary integrals.

ICHINT = i defines the type of boundary integral to be computed. The following values for ICHINT are available:

1. $\int\limits_{\partial\Omega} fuds$, where u denotes the solution defined by the input vector (VECTOR `%name` in the input block STRUCTURE) and f a function defined by ICHFUN.
2. $\int\limits_{\partial\Omega} f\mathbf{u}\cdot\mathbf{n}ds$, where **u** denotes the solution defined by the input vector (VECTOR i in the input block STRUCTURE) and **n** the normal defined at the boundary. If is supposed that the solution can be considered as a vector, which means that there are at least NDIM (dimension of space) degrees of freedom per point to be integrated.
3. $\int\limits_{\partial\Omega} f\mathbf{u}\cdot\mathbf{t}ds$, where **t** defines the tangential vector.

The default value for ICHINT is 1.

ICHFUN = j defines how the function f must be computed. The following values for ICHFUN are permitted:

0 The function f is identical to 1.

>**0** The function f must be computed by a function subroutine FUNC or CFUNC as described in the SEPRAN introduction Section 5.5.3. If the solution vector is complex CFUNC is used otherwise FUNC should be used. The value of ICHFUN is used as parameter ICHOIS in the input of the function subroutines.
At this moment ICHFUN>0 is only permitted in combination with ICHINT=1.

The default value for ICHFUN = 1.

IRULE = k defines the type of numerical integration rule to be applied. The following values of IRULE are available:

1. Trapezoid rule (Integration based upon two points)
2. Trapezoid rule with axi-symmetric co-ordinates, i.e. $ds = 2\pi r ds\prime$.
3. Simpson rule (Integration based upon three points)
4. Simpson rule with axi-symmetric co-ordinates, i.e. $ds = 2\pi r ds\prime$.

The default value for IRULE = 1.

CURVES (C1, C2, C3, ...) defines over which curves the integral must be computed. If a curve must be integrated in reversed direction, the curve number must be provided with a minus sign. Of course this possibility makes only sense for ICHINT > 1.

DEGREE_OF_FREEDOM = d defines which unknown in each point from the solution vector (VECTOR `%name`) is used.
When **u** is a vector (ICHINT>1), the degrees of freedom u_1, u_2 and u_3 in each nodal point are supposed to be the degrees of freedom d, $d+1$ and $d+2$ respectively.

END defines the end of the input block

5.4.11 The main keyword STRUCTURE

The block defined by the main keyword STRUCTURE defines which actions should be performed by program SEPCOMP. In fact this block defines the complete structure of the main program.

STRUCTURE should only be used if the standard options for the solution of a linear problem or non-linear problem do not suffice. In the block STRUCTURE it is precisely described which vectors and scalars are created, how they are created and in which sequence. STRUCTURE contains a number of commands which internally refer to separate subroutines. Each of these subroutines requires input. The input for these specific subroutines is defined in separate input blocks. Each of these blocks may be provided with a local sequence number.
The commands in STRUCTURE may refer to these sequence numbers. The block defined by the main keyword STRUCTURE starts with the command STRUCTURE at a separate record and ends with the keyword END on another separate record. In between commands may be given in any sequence and on separate records. However, the commands itself are carried out in exactly the sequence as given in this block. This means that the user himself is responsible for the correctness of the sequence of the commands. The only check that is performed is that vectors and scalars that are used as input have already been filled before.

STRUCTURE makes it possible to work with a number (100) of vectors (solutions and so on) as well as a number (1000) of scalars.
Each of them has a sequence number.
However, to increase readability we shall not use the sequence numbers of the vectors and scalars, but instead we use the names as defined in the input block CONSTANTS, subparts VARIABLES and VECTOR_NAMES. In the sequel the vector with name `name` will be denoted by `V%name` and the scalar with name `scalarname` will be denoted by `V%scalarname`. The reference to a vector or scalar must always be preceded by the % sign.

The block STRUCTURE consists of a series of commands that may be repeated. The following types of commands may be used in the block STRUCTURE:
(options are indicated between the square brackets "[" and "]"):

```
STRUCTURE
  PRESCRIBE_BOUNDARY_CONDITIONS [sequence_number = s] [vector = %name]
  SOLVE_LINEAR_SYSTEM  [seq_solve = s]  [vector = %name]
  SOLVE_NONLINEAR_SYSTEM [sequence_number = s]  [vector = %name]
  CREATE_VECTOR [sequence_number = s] [vector = %name]
  DERIVATIVES [seq_deriv = s] [vector = %name]
  INTEGRAL [seq_integral = i] [vector = %name] [scalar %scalarname ]
  BOUNDARY_INTEGRAL,[seq_boun_integral = i] [vector = %name] \\
                  [scalar1= %scalarname1 ] [scalar2= %scalarname2 ]
  OUTPUT [sequence_number = s] [vector = %name]
  COMPUTE_SCALAR %scalarname [options]
  SCALAR j = value or (FUNC=k)
  PRINT_scalar %scalarname [text='some text']
  PRINT_VECTOR %name [options]
  PRINT_TEXT, 'text between quotes'
END
```

Mark that the input file is case insensitive except for texts between quotes. Hence the use of capitals in the previous part is only to emphasize the commands.

Commands may be repeated and given in any order. However, they are executed in exactly the sequence given in the block which means that this sequence defines the complete program and hence must be logical. So it is for example necessary to prescribe the boundary conditions first and then to solve the system of linear equations, since otherwise the effect of the essential boundary conditions to the solution is not present and the solution may be undefined.

These commands have the following meaning:

STRUCTURE (mandatory) This keyword indicates the start of the input block STRUCTURE. All records following it until the record END is found define the complete structure of the program.

PRESCRIBE_BOUNDARY_CONDITIONS [sequence_number = s] [vector = %name]
With this command the vector `V%name` is provided with essential boundary conditions as described in the input block "ESSENTIAL BOUNDARY CONDITIONS" with sequence number s. If `V%name` already exists the values of `V%name` are changed, otherwise `V%name` is set equal to zero before applying the essential boundary conditions.
If sequence_number = s is omitted implicitly the next sequence number is assumed. Hence in the first "call" of prescribe_boundary_conditions sequence number 1 and so on.
The result of this operation is that the vector `V%name` has been filled or changed.

SOLVE_LINEAR_SYSTEM [seq_solve = s] [vector = %name]
The command solve_linear_system performs actually two independent steps.
Firstly the matrix and right-hand-side vector is built. Finally the system of linear equations is solved by the linear solver. Information about the solution process is read in the input block "SOLVE" with sequence number s as indicated by seq_solve = s.
Before applying the command solve_linear_system it is necessary that the essential boundary conditions have already been filled into the solution vector `V%name`. This may be done in several ways:

- By applying the command prescribe_boundary_conditions to `V%name`
- By applying the command create_vector to `V%name`
- By creating `V%name` through another operation like a previous solve.

`V%name` must be of the type solution vector.
The result of the total operation is that `V%name` has been filled with the solution of a linear differential equation.

SOLVE_NONLINEAR_SYSTEM [sequence_number = s] [vector = %name]
The command solve_nonlinear_system is comparable to the command
solve_linear_system. However, in this case a non-linear system of equations is solved by an iteration process. In each step of the iteration process, systems of equations are built and a system of linear equations is solved.
Before applying the command solve_nonlinear_system it is necessary that at least the essential boundary conditions have already been filled into the solution vector `V%name`. Usually the iteration process expects that a complete initial estimate has been filled in `V%name`. `V%name` may be filled in the same way as described for the linear problems.
The result of this operation is that `V%name` has been filled with the solution of a non-linear differential equation.

The various options have the following meaning:

sequence_number = *s* refers to information about the coefficients for the differential equation and natural boundary conditions. This information is given in the input block "NONLINEAR EQUATIONS" with sequence number s. This block also contains information about the linear solver to be applied. If sequence_number = s is omitted implicitly the next one is assumed.

vector = %name refers to the vector that is computed by this command. If omitted the first vector in the block VECTOR_NAMES is used.

CREATE_VECTOR [sequence_number = s] [vector = %name]
The command create_vector may be used to create the vector `V%name` explicitly. This vector may be used as initial estimate for a non-linear problem, or to prescribe the essential boundary

conditions.
The definition of the vector to be created is given in the input block "CREATE" with sequence number s. If sequence_number = s is omitted the next one is used.
If vector = %name is omitted the first vector is assumed.
The result of this operation is that a vector `V%name` has been created.

DERIVATIVES [seq_deriv = s] [vector = %name]
The command derivatives may be used to create the vector `V%name` as derived quantity of previously constructed vectors.

Input concerning the derived quantities to be computed is defined in the input block "DERIVATIVES" with sequence number s.
The result of this operation is that a vector `V%name` has been created.

INTEGRAL [seq_integral = i] [vector = %name] [scalar %scalarname]
The command integral may be used to compute scalar S%scalarname as integral over vector `V%name`. The result of this operation is that the scalar S%scalarname has got a value.

BOUNDARY_INTEGRAL ,[seq_boun_integral = i], [vector=%name], [scalar1=%scalarname1], [scalar2=%scalarname2]
The command boundary_integral may be used to compute scalar %scalarname1 as an integral of VECTOR %name over (a part of) the boundary. If vector = %name is omitted the first vector is assumed
Input concerning the boundary integral to be computed is defined in the input block "BOUNDARY_INTEGRAL" with sequence number s. If s is omitted the next one is assumed.
If the integral to be computed is a vector then the second component is stored in SCALAR %scalarname2. The result of this operation is that the scalar S%scalarname1 has got a value and possibly the scalar S%scalarname2 too.

OUTPUT [sequence_number = s] [vector = %name]
The vector `V%name` and, depending on the definition in the input block "OUTPUT", the next vectors, are written to the file sepcomp.out for post-processing purposes.
Information about what output should be written must be stored in the input block "OUTPUT" with sequence number s. If s is omitted the next one is assumed.

COMPUTE_SCALAR %scalarname [options]
The command compute_scalar %scalarname computes S%scalarname by manipulation of vectors. Which vectors are manipulated and how is defined by the options. The following options are available:

```
NORM = j, VECTOR %name [degfd k]
NORM_DIF = j, VECTOR1 = %name1, VECTOR2 = %name2 [degfd k]
AVERAGE VECTOR %name [degfd k]
```

These options have the following meaning:

NORM computes S%scalarname as norm of V%name. The value of j defines the type of norm to be used. Possible values:

1. $\| \mathbf{u} \| = \sum\limits_{i=1}^{N} | u_i |$
2. $\| \mathbf{u} \| = \left(\sum\limits_{i=1}^{N} u_i^2 \right)^{\frac{1}{2}}$
3. $\| \mathbf{u} \| = \max\limits_{1 \le i \le N} | u_i |$
4. $\| \mathbf{u} \| = \frac{\sum_{i=1}^{N} |u_i|}{N}$

5. $\| \mathbf{u} \| = \sqrt{\sum\limits_{i=1}^{N} u_i^2/N}$

If DEGFD k is given only the $DEGFD^{th}$ degree of freedom per point is taken into account.

NORM_DIF computes S%scalarname as norm of V%name1 - V%name2. The value of j defines the type of norm to be used. The same values of j as for the option NORM are available. If DEGFD k is given only the $DEGFD^{th}$ degree of freedom per point is taken into account.

AVERAGE computes S%scalarname as the average value of the vector V%name. If DEGFD k is given only the $DEGFD^{th}$ degree of freedom per point is taken into account.

SCALAR %scalarname = *choice*
The command SCALAR %scalarname sets S%scalarname equal to the value defined by *choice*. The following possibilities for *choice.* are available:

```
value
(FUNC=k)
MIN ( F1, F2, F3, ... )
MAX ( F1, F2, F3, ... )
```

These options have the following meaning:

If a value is given explicitly, the scalar gets this value.

FUNC=k gives the scalar the value FUNCSCAL (k, SCALARS), which means that it may be a function of the other scalars.
See Section 5.6.7 for a description of FUNCSCAL.

min or max (F1,F2,F3,...) means that the scalar gets the minimum respectively maximum value of F1, F2, F3 and so on. F1, F2, F3, .. may be either numbers or of the shape Sj, referring to scalar j. Of course scalar j must have been given a value before.

PRINT_SCALAR %scalarname [text='some text']
The command PRINT_SCALAR prints the value of S%scalarname to the output file. If text is given it should be followed by some text between quotes. This text is used to identify the scalar to be printed in the following way:

```
text = S%scalarname
```

where S%scalarname denotes the value of S%scalarname.

PRINT_VECTOR %name [options]
The command PRINT_VECTOR prints the value of V%name to the output file. The following options are available:

```
text = 't'
curves = c1, c2, cn, ...
```

These options have the following meaning:

text should be followed by some text between quotes. This text is used to identify the vector to be printed.

curves followed by Ci, Cj, Ck, ... ensures that the printing of the solution is restricted to the curves given in the list.

Remarks:

PRINT_TEXT , 'text between quotes'
The command PRINT_TEXT prints the text between the quotes to the output file.

END (mandatory) Indicates the end of the STRUCTURE block.

If the block STRUCTURE is omitted SEPCOMP checks for the presence of the block NONLINEAR EQUATIONS. If this block is available SEPCOMP reacts as if the block STRUCTURE is available with the following contents:

```
structure
   prescribe_boundary_conditions, sequence_number = 1
   solve_nonlinear_system, sequence_number = 1
   output, sequence_number = 1
end
```

Otherwise it is supposed that a linear system must be solved and the structure is:

```
structure
   prescribe_boundary_conditions, sequence_number = 1
   solve_linear_system, seq_solve = 1
   output, sequence_number = 1
end
```

5.5 Description of some function subroutines to be used together with program SEPCOMP

The user may provide data to program SEPCOMP by the input file. However, if coefficients, boundary conditions and so on depend on space, this is not a practical possibility. In those cases it is much easier to give data directly in the form of a function. At this moment SEPRAN does not allow the input of functions through the input file. For that reason the user must give space dependent data by a so-called function subroutine.
In this section the user interface of some of these function subroutines is given. Which of the subroutines is used depends on the input in the standard input file. In the specific parts it is indicated which function subroutine is required for special data.

In 5.5.1 function subroutine FUNCBC is described for the definition of real essential boundary conditions.
5.5.2 describes subroutine CFUNCB for complex essential boundary conditions.
5.5.3 deals with function subroutines FUNC and CFUNC, which are used to define complete real or complex fields.

5.5.1 Function subroutine FUNCBC

Description

With this function subroutine a function may be defined, for the creation of real boundary conditions.
FUNCBC must be written by the user.

Heading

```
function funcbc ( ifunc, x, y, z )
```

Parameters

DOUBLE PRECISION FUNCBC, X, Y, Z
INTEGER IFUNC

IFUNC Choice parameter. This parameter enables the user to distinguish between several cases. IFUNC is defined by the user in the input part ESSENTIAL BOUNDARY CONDITIONS. Its value is equal to the parameter i in FUNC=i.

X,Y,Z X, y and z-coordinates of the nodal point. For each nodal point this subroutine is called.

FUNCBC FUNCBC should get the computed value of the function in the nodal point.

Input

IFUNC, X, Y, and Z have been filled by SEPCOMP depending on the dimension of the space.

Output

FUNCBC must have a value.

Example

Suppose for IFUNC = 1 the function f(x,y)=xy, and for IFUNC = 2 the function f(x,y)=sin(x) is required.
Then FUNCBC can be programmed as follows:

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

c     --- see the remarks in 2.5

      if ( ifunc .eq. 1 ) then

c     --- ifunc = 1   f = x y

         funcbc = x * y
      else

c     --- ifunc = 2   f = sin ( x )

         funcbc = sin(x)
      endif
      end
```

5.5.2 Subroutine CFUNCB

Description

With this subroutine the values of the boundary conditions may be defined, when complex arithmetic is used.
CFUNCB must be written by the user.

Heading

```
subroutine cfuncb (ifunc, x, y, z, comval)
```

Parameters

DOUBLE PRECISION X, Y, Z

COMPLEX ∗16 COMVAL

INTEGER IFUNC

IFUNC Choice parameter. This parameter enables the user to distinguish between several cases. IFUNC is defined by the user in the input part ESSENTIAL BOUNDARY CONDITIONS. Its value is equal to the parameter *i* in FUNC=*i*.

X,Y,Z X, y and z-coordinates of the nodal point. For each nodal point this subroutine is called.

COMVAL Complex output parameter. COMVAL must be given a value by the user.

Input

ICHOIS, X, Y and Z have got a value depending on the dimension of the space.

Output

COMVAL must have been filled by the user.

Example

Suppose for IFUNC=1 the function f(x,y)=(x,y) is required. Then CFUNCB may be programmed as follows:

```
subroutine  cfuncb ( ifunc, x, y, z, comval )
implicit none
integer ifunc
double precision x, y, z
complex *16 comval
if ( ifunc .eq .1 ) then
        comval = dcmplx ( x, y )
else
     .
     .
     .
endif
end
```

Remark

The FORTRAN intrinsic function DCMPLX combines two real variables into one complex variable.

5.5.3 Function subroutines FUNC and CFUNC

Description

With these function subroutines a function may be defined.
FUNC and CFUNC must be written by the user. In the case of real vectors FUNC and in the case of complex vectors CFUNC must be used.

Heading

```
function func (ifunc, x, y, z)
```

or

```
function cfunc (ifunc, x, y, z)
```

Parameters

DOUBLE PRECISION FUNC, X, Y, Z

INTEGER IFUNC

COMPLEX $*$ 16 CFUNC

IFUNC Choice parameter. This parameter enables the user to distinguish between several cases. IFUNC has been given a value by program SEPCOMP.

X, Y, Z X, y and z-co-ordinates of the nodal point. For each nodal point this subroutine is called.

FUNC should get the computed value of the function in the nodal point. (Real case only)

CFUNC should get the computed value of the function in the nodal point. (Complex case only)

Input

IFUNC, X, Y and Z have got a value depending on the dimension of the space.

Output

FUNC or CFUNC has got a value.

For an example of how to write FUNC or CFUNC see FUNCBC (5.5.1).

5.6 How to program your own element subroutines

In the Numerical Analysis Lab the student must program his own elements. Of course most of the exercises can be made with standard SEPRAN element subroutines using type numbers larger than 99. However, programming your own element is the main goal of the lab.
It is always necessary to program your own element subroutine ELEMSUBR, that is to be used both in the case of a linear and of a non-linear problem. See Section 5.6.1 for a description.
If derived quantities like a first derivative must be computed it is also necessary to program an element subroutine ELDERVSUBR. See Section 5.6.2 for a description.
If integrals must be computed it is necessary to program an element subroutine ELINTSUBR. See Section 5.6.3 for a description.
The Sections 5.6.4, 5.6.5 and 5.6.6 describe some help subroutine to simplify the printing of arrays.
The general idea is the following:

If the large matrix is built a subroutine BUILD is called that makes a loop over all elements. For each element the element subroutine ELEMSUBR is called. This subroutine is supposed to compute the element matrix and element vector. BUILD then adds this element matrix and element vector to the large matrix and vector in the right positions.
In the same way a loop over the element subroutines is performed for the integration and derivative subroutines.

5.6.1 Subroutine ELEMSUBR

Description

Subroutine ELEMSUBR is called by a subroutine BUILD which builds the large matrix and vector. This is used both for the linear problems as for non-linear problems.
This subroutine is only used for type numbers between 1 and 99, hence for the Numerical Analysis Lab this subroutine is obliged.

Use the command `sepgetpract` to get the correct interface in your local directory. See Section 5.2

The general structure of subroutine BUILD is as follows:

```
clear large matrix and large vector
For all element groups and all boundary element groups do
   For all elements in the group do
      call ELEMSUBR
      add element matrix and element vector to large matrix and large vector
   end_For
end_For
```

Heading

```
      subroutine elemsubr ( ndim, npelm, x, nunk_pel, elem_mat,
     +                   elem_vec, elem_mass, uold, itype )
```

Parameters

INTEGER NDIM, NPELM, NUNK_PEL, ITYPE

DOUBLE PRECISION X(NPELM,NDIM), ELEM_MAT(NUNK_PEL,NUNK_PEL), ELEM_VEC(NUNK_PEL), ELEM_MASS(NUNK_PEL), UOLD(NUNK_PEL)

NDIM (input parameter)
Defines the dimension of the space in which the problem is solved. For nearly all problems in the lab ndim = 2.

NPELM (input parameter)
Defines the number of points in the element. So for a linear triangle NPELM = 3, and for a linear boundary element NPELM = 2.

X (input array)
Double precision two-dimensional array of size NPELM $\times$ NDIM. X(i,1) contains the x-coordinate of the i^{th} node in the element and X(i,2) the y-coordinate of this node.
Mark that it concerns the local numbering of the element, not the global node numbers.

NUNK_PEL (input parameter)
Defines the number of degrees of freedom in the element.
Usually this number is equal to NPELM, but for example, if the number of degrees of freedom per point is 2, it is 2 $\times$ NPELM.

ELEM_MAT (output array)
In this double precision two-dimensional array the student must store the element matrix, in the following way:

ELEM_MAT(i,j) = s_{ij} ; i,j = 1(1)NUNK_PEL.

The degrees of freedom in an element are stored sequentially, first all degrees of freedom corresponding to the first point, then to the second, etcetera.
The local sequence of the nodes is defined by Table 4.1.1.

ELEM_VEC (output array)
In this double precision array the student must store the element vector, in the following way:
ELEM_VEC(i) = f_i ; i = 1(1)NUNK_PEL.

ELEM_MASS (output array)
In this double precision two-dimensional array the student must store the element mass matrix, provided the mass matrix must be computed, in the following way:
ELEM_MASS(i,j) = s_{ij} ; i,j = 1(1)NUNK_PEL.
This matrix should only be filled if a mass matrix is required, for example for time-dependent problems.

UOLD (input array)
In this array the old solution, as indicated by V1, is stored. This solution may contain the boundary conditions only, if the array has been created by prescribe_boundary_conditions, but also a starting vector if V1 has been created by create or even the previous solution in an iteration process if nonlinear_equations is used.
The sequence in which UOLD is filled is the same as used in X and ELEM_MAT. Hence first all degrees of freedom for the first local point, then for the second one and so on.
This array is only used in case of non-linear problems.

ITYPE (input parameter)
This parameter defines the type number of the element. This type number has been defined in the input block PROBLEM as part of the statements:

```
ELGRP i = (type = n3)
BNGRP 1 = (type = n1)
```

The student may utilize ITYPE to distinguish between different types of element matrices, for example to distinguish between internal elements and boundary elements.

Input

Program SEPCOMP (subroutine BUILD) fills the arrays X and array UOLD before the call of ELEMSUBR.
Also the parameters NDIM, NPELM, NUNK_PEL and ITYPE have got a value.

Output

The student must fill the arrays ELEM_MAT, ELEM_VEC and in case of time-dependent problems ELEM_MASS.

Interface

Subroutine ELEMSUBR must be programmed as follows:

```
      subroutine elemsubr ( ndim, npelm, x, nunk_pel, elem_mat,
     +                      elem_vec, elem_mass, uold, itype )
      implicit none
      integer ndim, npelm, nunk_pel, itype
      double precision x(npelm,ndim), elem_mat(nunk_pel,nunk_pel),
     +                 elem_vec(nunk_pel), elem_mass(nunk_pel),
     +                 uold(nunk_pel)

c     --- declarations of local variables
c         for example:
      integer i, k

      if ( itype.eq.1 ) then

c     --- statements to fill the arrays elem_mat and elem_vec

         do k = 1, nunk_pel
            do i = 1, nunk_pel
               elem_mat(i,k) = "s(ik)"
            end do
            elem_vec(k) = "f(i)"
         end do

      else if ( itype.eq.2 ) then

c     --- the same type of statements for itype = 2, etcetera

      end if
      end
```

Remarks

- For problems in complex variables (like the Helmholtz equation) one may declare ELEM_MAT and ELEM_VEC as complex *16 arrays, which means that they are treated as double precision complex arrays.
- Almost all the errors that are made by students are in the subroutine ELEMSUBR. Since debugging of Fortran programs goes beyond the goal of the lab, it is advised to use print statements in the element subroutine to detect errors.
  For example to print the value of a variable `var` use

  ```
  print *, 'var = ', var
  ```

  To print the contents of a double precision array for example the element vector the SEPRAN subroutine PRINTREALARRAY may be used, see Section 5.6.4, to print the contents of an integer array: PRINTINTEGERARRAY, see Section 5.6.5. To print the contents of the element matrix use PRINTMATRIX, see Section 5.6.6.

  These subroutines may be for example used as follows:

  ```
   subroutine elemsubr ( ndim, npelm, x, nunk_pel, elem_mat,
  +                      elem_vec, elem_mass, uold, itype )
   implicit none
         .
         .
         .
         .
   call printrealarray ( elem_vec, nunk_pel, 'element vector' )
   call printmatrix ( elem_mat, nunk_pel, 'element matrix' )
   end
  ```

5.6.2 Subroutine ELDERVSUBR

Description

Subroutine ELDERVSUBR is called by a subroutine DERIV which builds a large vector by averaging over adjacent elements.
This subroutine is only used for type numbers between 1 and 99. It is called if and only if the option derivatives is used in the STRUCTURE block.
The general structure of subroutine DERIV is as follows:

```
clear large vector and weight vector
For all element groups do
   For all elements in the group do
      call ELDERVSUBR
      add element vector and element weight vector to large vector
          and weight vector
   end_For
end_For
Divide large vector by weight vector (element wise)
```

Heading

```
      subroutine eldervsubr ( ndim, npelm, x, nunk_pel, elem_vec,
     +                     elem_weight, uold, itype, icheld )
```

Parameters

INTEGER NDIM, NPELM, NUNK_PEL, ITYPE, ICHELD

DOUBLE PRECISION X(NPELM,NDIM), ELEM_VEC(NUNK_PEL), ELEM_WEIGHT(NUNK_PEL), UOLD(NUNK_PEL)

NDIM (input parameter)
Defines the dimension of the space in which the problem is solved. For nearly all problems in the lab ndim = 2.

NPELM (input parameter)
Defines the number of points in the element. So for a linear triangle NPELM = 3, and for a linear line element NPELM = 2.

X (input array)
Double precision two-dimensional array of size NPELM × NDIM. X(i,1) contains the x-coordinate of the i^{th} node in the element and X(i,2) the y-coordinate of this node.
Mark that it concerns the local numbering of the element, not the global node numbers.

NUNK_PEL (input parameter)
Defines the number of degrees of freedom in the element.
Usually this number is equal to NPELM, but for example, if the number of degrees of freedom per point is 2, it is 2 × NPELM.

ELEM_VEC (output array)
In this double precision array the student must store the element vector, in the following way:

ELEM_VEC(i) = f_i ; i = 1(1)NUNK_PEL.

It concerns the derived quantity that must be computed.
The sequence that must be used in case of more unknowns per point, like for example when computing the gradient, is:
First all unknowns in the first point of the element, followed by all unknowns in the second point and so on.

ELEM_WEIGHT (output array)
In this double precision array the student must store the element weight vector, in the following way:
ELEM_WEIGHT(i) = w_i ; i = 1(1)NUNK_PEL.
This weight vector is used for averaging purposes. See *method*.
The weight vector may for example consists of elements that are all equal to 1 or elements that are all equal to the area of the element.
If you do not know what to choose, make ELEM_WEIGHT completely equal to 1.

UOLD (input array)
In this array the solution from which the derived quantities must be computed, as indicated by Vi, is stored.
The sequence in which UOLD is filled is the same as used in X and ELEM_VEC. Hence first all degrees of freedom for the first local point, then for the second one and so on.
In UOLD the value in the vertices are stored.

ITYPE (input parameter)
This parameter defines the type number of the element. This type number has been defined in the input block PROBLEM as part of the statements:

```
ELGRP i = (type = n3)
BNGRP 1 = (type = n1)
```

The student may utilize ITYPE to distinguish between different types of element matrices, for example to distinguish between internal elements and boundary elements.

ICHELD (input parameter)
This is the input parameter defined in the input block DERIVATIVES. This parameter may be used to distinguish between possibilities.

Input

Program SEPCOMP (subroutine DERIV) fills the arrays X and array UOLD before the call of ELDERVSUBR.
Also the parameters NDIM, NPELM, NUNK_PEL, ITYPE and ICHELD have got a value.

Output

The student must fill the arrays ELEM_VEC and ELEM_WEIGHT.

Interface

Subroutine ELDERVSUBR must be programmed as follows:

```
      subroutine eldervsubr ( ndim, npelm, x, nunk_pel, elem_vec,
     +                        elem_weight, uold, itype, icheld )
      implicit none
      integer ndim, npelm, nunk_pel, itype, icheld
      double precision x(npelm,ndim), elem_vec(nunk_pel),
     +                 elem_weight(nunk_pel), uold(nunk_pel)

c     ---  declarations of local variables
c          for example:
      integer i

      if ( itype.eq.1 ) then

c     --- statements to fill the arrays elem_vec and elem_weight

         do i = 1, nunk_pel
            elem_vec(i) = "f(i)"
            elem_weight(i) = 1
         end do

      else if ( itype.eq.2 ) then

c     --- the same type of statements for itype = 2, etcetera

      end if
      end
```

Method

The averaging procedure is as follows. Suppose that nodal point j is lying in K different elements. Let the quantity q be given in nodal point j, with a different value in each element. In order to compute an averaged value of q in j, weights w_i (i = 1, 2, . . . , K) for each element corresponding to nodal point j must be defined. The averaged value of q in nodal point j is computed by the following formula:

$$\bar{q}(x^j) = \frac{\sum\limits_{i=1}^{K} q_i(x^j)\ w_i}{\sum\limits_{i=1}^{K} w_i} w_i \ \geq \ 0; \qquad \sum_{i=1}^{K} w_i > 0 \tag{5.6.2.1}$$

with
$\bar{q}(x^j)$ the averaged value of q in nodal point j,
$q_i(x^j)$ the value of q in nodal point j with respect to element i,
w_i the weight corresponding to nodal point j with respect to element i.

Simple choices are for example:

$w_i \ = \ 1$ or
w_i = area of element i.

The adding process over the various elements is carried out by program SEPCOMP, it is sufficient to compute the derived quantities and weights with respect to each nodal element separately with the aid of subroutine ELDERVSUBR.

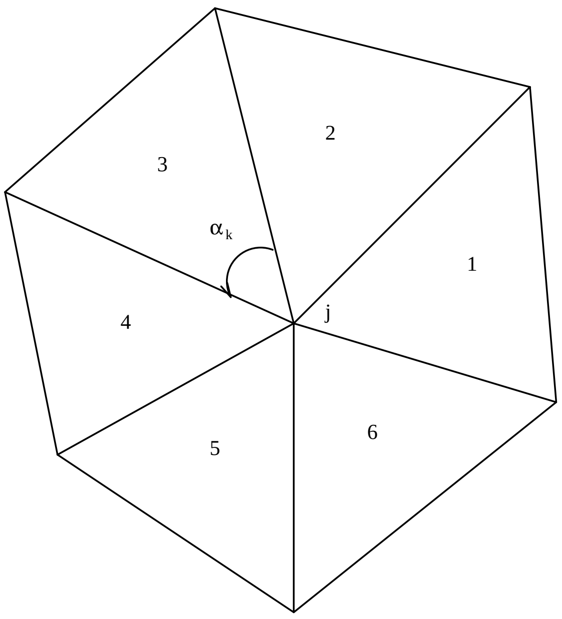


Figure 5.6.2.1: nodal point j in different elements

5.6.3 Function subroutine ELINTSUBR

Description

Function subroutine ELINTSUBR is called by a subroutine INTEGRAL which computes the integral over a region by adding integrals over elements.
This subroutine is only used for type numbers between 1 and 99. It is called if and only if the option integrals is used in the STRUCTURE block.
The general structure of subroutine INTEGRAL is as follows:

```
sum := 0
For all element groups do
   For all elements in the group do
      sum := sum + ELINTSUBR (...)
   end_For
end_For
```

Heading

```
      function elintsubr ( ndim, npelm, x, nunk_pel,
     +                     uold, itype, icheli )
```

Parameters

INTEGER NDIM, NPELM, NUNK_PEL, ITYPE, ICHELI

DOUBLE PRECISION X(NPELM,NDIM), ELINTSUBR, UOLD(NUNK_PEL)

ELINTSUBR (output parameter)
The student must give elintsubr the value of the integral over the element to be computed.

NDIM (input parameter)
Defines the dimension of the space in which the problem is solved. For nearly all problems in the lab ndim = 2.

NPELM (input parameter)
Defines the number of points in the element. So for a linear triangle NPELM = 3, and for a linear line element NPELM = 2.

X (input array)
Double precision two-dimensional array of size NPELM × NDIM. X(i,1) contains the x-coordinate of the i^{th} node in the element and X(i,2) the y-coordinate of this node.
Mark that it concerns the local numbering of the element, not the global node numbers.

NUNK_PEL (input parameter)
Defines the number of degrees of freedom in the element.
Usually this number is equal to NPELM, but for example, if the number of degrees of freedom per point is 2, it is 2 × NPELM.

UOLD (input array)
In this array the function to be integrated, as indicated by Vi, is stored.
The sequence in which UOLD is filled is the same as used in X. Hence first all degrees of freedom for the first local point, then for the second one and so on.

ITYPE (input parameter)
This parameter defines the type number of the element. This type number has been defined in the input block PROBLEM as part of the statements:

```
ELGRP i = (type = n3)
```

The student may utilize ITYPE to distinguish between different types of element matrices, for example to distinguish between internal elements and boundary elements.

ICHELI (input parameter)
This is the input parameter defined in the input block INTEGRAL. This parameter may be used to distinguish between possibilities.

Input

Program SEPCOMP (subroutine INTEGRAL) fills the arrays X and array UOLD before the call of ELINTSUBR.
Also the parameters NDIM, NPELM, NUNK_PEL, ITYPE and ICHELI have got a value.

Output

The student must give ELINTSUBR a value.

Interface

Function subroutine ELINTSUBR must be programmed as follows:

```
      function elintsubr ( ndim, npelm, x, nunk_pel,
     +                     uold, itype, icheli )
      implicit none
      integer ndim, npelm, nunk_pel, itype, icheli
      double precision x(npelm,ndim), uold(nunk_pel), elintsubr

c     ---  declarations of local variables

      if ( itype.eq.1 ) then

c     --- statements to compute the integral over the element

         elintsubr = ..

      else if ( itype.eq.2 ) then

c     --- the same type of statements for itype = 2, etcetera

      end if
      end
```

5.6.4 Subroutine PRINTREALARRAY

Description

Subroutine PRINTREALARRAY is a special subroutine that is meant to print double precision arrays inside a user written element subroutine.

Heading

```
subroutine printrealarray ( array, n, text )
```

Parameters

INTEGER N

DOUBLE PRECISION ARRAY(N)

CHARACTER ∗ (∗) TEXT

N (input parameter)
Defines the length of ARRAY.

ARRAY (input array)
Double precision array of size N to be printed.

TEXT (input parameter)
Text to be printed in the heading of the print.

Input

The parameters N and TEXT must have a value.
Array ARRAY must have been filled.

Output

The contents of array ARRAY are printed

5.6.5 Subroutine PRINTINTEGERARRAY

Description

Subroutine PRINTINTEGERARRAY is a special subroutine that is meant to print integer arrays inside a user written element subroutine.

Heading

```
subroutine printintegerarray ( iarray, n, text )
```

Parameters

INTEGER N, IARRAY(N)

CHARACTER ∗ (∗) TEXT

N (input parameter)
Defines the length of IARRAY.

IARRAY (input array)
Integer array of size N to be printed.

TEXT (input parameter)
Text to be printed in the heading of the print.

Input

The parameters N and TEXT must have a value.
Array IARRAY must have been filled.

Output

The contents of array ARRAY are printed

5.6.6 Subroutine PRINTMATRIX

Description

Subroutine PRINTMATRIX is a special subroutine that is meant to print two-dimensional double precision arrays inside a user written element subroutine.

Heading

```
subroutine printmatrix ( array, n, text )
```

Parameters

INTEGER N

DOUBLE PRECISION ARRAY(N,N)

CHARACTER $*$ **(**$*$**)** TEXT

N (input parameter)
Defines the length of ARRAY.

ARRAY (input array)
Two dimensional double precision array of size N $\times$ N to be printed.

TEXT (input parameter)
Text to be printed in the heading of the print.

Input

The parameters N and TEXT must have a value.
Array ARRAY must have been filled.

Output

The contents of array ARRAY are printed

5.6.7 Function subroutine FUNCSCAL

Description

The function subroutine FUNCSCAL is a user written subroutine that must be provided if the option SCALAR j, FUNC = k is used in the input block "STRUCTURE". It is used to construct a scalar as function of previously computed scalars.

Heading

```
function funcscal ( k, arscalars )
```

Parameters

DOUBLE PRECISION FUNCSCAL, ARSCALARS(∗)

INTEGER K

ARSCALARS In this array all scalars are stored that are defined in the part created by the input block "STRUCTURE". The scalars are stored in the sequence defined by the user, which means that S1 is stored in ARSCALARS(1), S2 in ARSCALARS(2) etcetera. The user must know himself which scalars are filled and which not.

K This parameter may be used to distinguish between various cases. K is identical to the parameter k given in FUNC = k. K has got a value by program SEPCOMP.

FUNCSCAL Result of the computation. The user must give FUNCSCAL a value as function of the other scalars.

Input

K has been given a value by SEPCOMP.
Array ARSCALARS has been filled by program SEPCOMP, at least for those values that have been explicitly computed by the user in the part STRUCTURE before the call SCALAR j, FUNC = k.

Output

FUNCSCAL must have a value

Example Suppose that S3 must be created as function of S1 and S2 in the following way:

$$S3 = sin(S1)cos(S2). \tag{5.6.7.1}$$

In the input block "STRUCTURE" the command

SCALAR 3, FUNC = 1 must be given and a function subroutine FUNCSCAL of the following shape must be provided by the user:

```
      FUNCTION FUNCSCAL ( K, ARSCALARS )
      IMPLICIT NONE
      DOUBLE PRECISION FUNCSCAL ARSCALARS
      INTEGER K

      if ( k.eq.1 ) then

c     --- Compute the result for k = 1

         funcscal = sin(arscalars(1)) * cos(arscalars(2))

      else

c     --- Other values of k

        .
        .

      end if

      END
```

6 The postprocessing part of SEPRAN

6.1 Introduction

In the post processing part of SEPRAN, the output of the solution and derived quantities is produced in a readable (visible) form. Integrals over quantities, integrals over boundaries etc. may be computed and printed or plotted. The output generated may be produced in either print or plot form. Print output is both written to the screen or to a file for later reproducing on a printer.

The post processing is performed by the main program SEPPOST. It requires two types of input.

First it uses some files produced by the mesh generation part (meshoutput) and the computational part (sepcomp.out and sepcomp.inf).

Secondly it requires input from the standard input file. (An interactive version will be available in due course).

The input of SEPPOST is described in the next paragraphs.

6.2 describes the general shape of the input, including the so-called compute commands, the define commands and the reset commands,

6.3 treats the various print commands,

6.4 the plot commands and

6.5 is devoted to some special commands with respect to time-dependence.

6.2 General input for program SEPPOST

The input for the post processing part must be opened with the keyword POSTPROCESSING and must be closed with the keyword END.

Structure of the input file:

```
postprocessing
   print v%name ....
   plot ....
   compute v%name_1 ....
   define  ....
   reset ....
   time = ....
   time history ....
end
```

The actual post processing commands may be given in any order, with the restriction that vectors V%name to be printed or plotted must have been defined before, for example by a compute statement.
Vector names used in the computational program are also known in SEPPOST. Vector names that are new, for example the ones used in a compute statement, must be defined in the block CONSTANTS, subpart VECTOR_NAMES.
One can consider this as a kind of declaration.

The PRINT commands are treated in 6.3, the PLOT commands in 6.4 and the TIME (HISTORY) commands in 6.5.

DEFINE and RESET commands

The DEFINE and RESET commands are used to set or reset of some defaults for printing or plotting. Their general syntax is:

```
  define plot parameters = ....
  define colour table = ....
  reset plot parameters
  reset colour table
```

With the define plot parameters statement, the user defines new defaults for the plot parameters. These defaults remain valid until the user resets plot parameters with the reset command, or a new define plot parameters is read. For a description of the plot parameters the user is referred to 6.4.
Remark: one of the plot parameters: region = (xmin, xmax, ymin, ymax) is also used for the print commands. So if this parameter is also given in the define plot parameters, it affects the print output.

The statement define colour table defines the colour numbers for coloured plots. See 6.4.

COMPUTE commands

The COMPUTE command is used to define a vector V%name_1 as function of an already available vector V%name. Using the same name %name_1 in a new COMPUTE statement redefines vector V%name_1.
The general syntax for the compute statements is:

```
compute V%name_1 = stream function V%name [, start node =s]//
                   [, stream function value = f]
compute V%name_1 = velocity profile V%name [,degfd=k] origin=(O_x, O_y)]//
                   [,angle = a ]
compute V%name_1 = intersection V%name [,degfd=k] origin=(O_x, O_y)]//
                   [,angle = a ]
```

Meaning of these commands:

compute V%name_1 = stream function V%name means that vector V%name_1 must be computed as stream function from the velocity vector V%name. V%name is supposed to be a vector with the two x and y velocity components as first and second component in each nodal point. If start node = s is given, the value of the stream function is set in nodal point s (default s=1).
stream function value = f defines the value in the start node s (default f=0).

compute V%name_1 = velocity profile V%name defines vector V%name_1 as a function given by one of the velocity components (degfd=k, default k=1) along the line with origin (O_x, O_y) (default (0,0)) under an angle of a degrees (default a=0). This possibility is only permitted for two-dimensional vector fields. The intersection of the line with the mesh is computed and the solution is interpolated onto this line. If V%name is complex, V%name_1 is complex too.

Remark: at this moment the method is sensitive to round off, which means that if a line coincides with the boundary of the mesh, only some parts or no part at all may be found in the intersection. In that case it is recommended to shift the line over a small distance.

compute V%name_1 = intersection V%name defines vector V%name_1 as a function given by the component k of the solution V%name along the line with origin (O_x, O_y) (default (0,0)) under an angle of a degrees (default a=0). k is defined by degfd=k (default k=1). This possibility is only available for functions defined on a two-dimensional mesh. Furthermore this possibility is completely identical to the preceding one, including the remark given before.

Compute statements only define the vector V%name_1, which means that the actual computation is performed only if necessary. At most 26 vectors are allowed in SEPPOST.

6.3 Print commands for program SEPPOST

The general input for the program SEPPOST is described in 6.2. This paragraph is devoted to the available print commands.

At this moment only one print command is available. The syntax of the print commands is:
Options are indicated between the square brackets [and].

```
PRINT V%name [,sequence = (y)] [, region = (xmin, xmax, ymin, ymax) ]
```

PRINT V%name = prints the complete vector, together with the corresponding nodal point numbers and the co-ordinates.
If no sequence is given the co-ordinates are ordered in increasing x-sequence and for constant x-value in increasing y-sequence.

sequence = (y) means first increasing y-sequence and then increasing x-sequence.

region = $(xmin, xmax, ymin, ymax)$ indicates that only the the points with co-ordinates in the range of $xmin \le x \le xmax$ and $ymin \le y \le ymax$ are printed or plotted. If $ymin, ymax$ is omitted, then the complete y-range is used.

The region to be printed may also be defined with the statement
DEFINE PLOT PARAMETERS region = $(\ldots\ldots)$ which affects both plots and prints.

6.4 PLOT commands for program SEPPOST

The general input for the program SEPPOST is described in 6.2. This paragraph is devoted to the available plot commands.

The syntax of the plot commands is:
Options are indicated between the square brackets [and].

```
PLOT CONTOUR V%name [,degfd = k] [plot parameters] [nlevel = n] //
    [levels = (q1,q2, ...)] [minlevel = min] [maxlevel = max] //
    [smoothing factor = s]
PLOT VECTOR V%name [degfd1 = k_1 ,degfd2 = k_2] [plot parameters]
PLOT COLOURED LEVELS V%name [degfd = k] [plot parameters]//
    [nlevel = n] [levels = (q1,q2, ...)] [minlevel = min] [maxlevel = max]
PLOT FUNCTION V%name  [,plot parameters]
PLOT VELOCITY PROFILE V%name [degfd=k] [plot parameters]//
    [origin = (O_x , O_y)] [angle = a]
3D PLOT V%name  [plot parameters] [lindirec=l] [nstep=n]
PLOT BOUNDARY FUNCTION V%name, curves (C1, C2, C3, C5, ... ,Cn ) //
    [plot parameters]
```

PLOT CONTOUR indicates that contour lines (lines with constant function value) are plotted for the given function.
If degfd=k is given, then the k^{th} degree of freedom in each node is used as definition of the function, otherwise the first degree of freedom is used. When V%name is complex degfd = $2k$-1 refers to the real part of the k^{th} degree of freedom and degfd = $2k$ to the imaginary part of this degree of freedom.
The user may define the number of contour levels either by prescribing nlevel = n or by giving the contour levels explicitly through levels = $(q1, q2, \ldots)$. The default number of levels is 11. Besides prescribing the contour levels explicitly, the minimum and/or maximum level may also be given. If omitted, they are computed by the program.
The smoothing factor defines the kind of smoothing that must be applied to the contour lines. $s = 0$ (default), means no smoothing, the contour lines are piecewise linear. $s = 1$, computes a mean value between three succeeding values to filter some of the possible wiggles (Shuman filtering). For s = 2, 3, 4 and 5 a smooth spline is used to plot the contour lines. The higher the value of s, the smoother the spline. Although these pictures are much nicer for publication, the actual plot is in no way better than that of the non-smooth contours. Values larger than 5 are not permitted for s.

PLOT VECTOR V%name makes a vector plot of two of the degrees of freedom in each point. These components may be defined by degfd1 = k_1, degfd2 = k_2 respectively. If omitted degfd1 = 1, and degfd2 = 2 is assumed. With respect to complex V%name, see PLOT CONTOUR.

PLOT COLOURED LEVELS V%name makes a coloured contour plot of the array V%name, where the region between two levels is coloured.
If degfd=k is given, then the k^{th} degree of freedom in each node is used as definition of the function, otherwise the first degree of freedom is used. With respect to complex V%name, see PLOT CONTOUR.
The user may define the number of contour levels either by prescribing nlevel = n or by giving the contour levels explicitly through levels = $(q1, q2, \ldots)$. The default number of levels is 11. Besides prescribing the contour levels explicitly, the minimum and/or maximum level may also be given. If omitted, they are computed by the program.
The colours used for the plotting are the standard colours defined for your system. These colours may be changed by the statement define colour table. See colour table.

PLOT FUNCTION V%name makes a plot of a one dimensional function. At this moment only vectors defined by COMPUTE V%name = velocity profile or COMPUTE V%name =

intersection, (See 6.2) may be plotted by this command. If the solution corresponds to a one-dimensional mesh, the complete solution is plotted. If V%name is complex, degfd = 1 plots the real part and degfd = 2 the imaginary part.

PLOT VELOCITY PROFILE V%name_1 combines the commands COMPUTE V%name_1 = velocity profile V%name_1 as described in 6.2 and the command PLOT FUNCTION V%name.

3D PLOT V%name makes a three-dimensional plot with hidden lines of a function defined on a two-dimensional mesh. With the parameter LINDIREC the user indicates in which direction the surface lines are drawn. Possibilities:

1 parallel to y-axis

2 parallel to x-axis

3 lines are drawn in both directions

5 A series of three pictures with the options 1, 2 and 3 is made

The default value for LINDIREC is 3. NSTEP=n indicates 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. The default value for n is 1. With respect to complex V%name, see PLOT CONTOUR.

PLOT BOUNDARY FUNCTION V%name, CURVES (C1,. . ., Cn) may be used to plot a function defined along the curves C1 to Cn, where it is necessary that the end point of the i^{th} curve, is identical to the initial point of the $i+1^{th}$ curve. If negative curve numbers are used, the corresponding curve is used in reversed direction.

Plot parameters

The following plot parameters may be used at the place formally indicated by [,plot parameters]:

```
region = (xmin, xmax, ymin, ymax)
length = l
yfact = y
symbol = s
textx = ‘ ... ’
texty = ‘ ... ’
rotate
norotate
scales = (x_under, x_upper, y_under, y_upper)
number format = (n_x, m_x, n_y, m_y)
steps = (stepx, stepy)
factor = f
pict i of n
angle = alpha
```

These options may be separated by commas.

region = $(x\min, x\max, y\min, y\max)$ is used to define a cut of a two-dimensional region.

length = l gives the length of the plot in centimeters. Instead of length also plotfm may be used. The default length is machine dependent but usual values are 20 cm or 15 cm.

yfact = y Scale factor; all y-coordinates are multiplied by y before plotting the mesh. $y \neq 1$ should be used when the co-ordinates in x and y direction are of different scales, and hence the picture becomes too small. Default value: 1.

symbol = s defines the number of the symbol to be used for plotting a one-dimensional function (installation dependent).

textx = '...', texty = '... ' define the texts to be plotted along the axes (default x and y). The part between the quotes is used as text.

rotate means that the picture is rotated over an angle of 90°.

norotate means that the picture is not rotated.
Default: depending on the size of the picture.

scales = $(x_{under}, x_{upper}, y_{under}, y_{upper})$ define the range of the scales along the axis of a one-dimensional plot (See Figure 6.4.1). (Default: computed by the program).

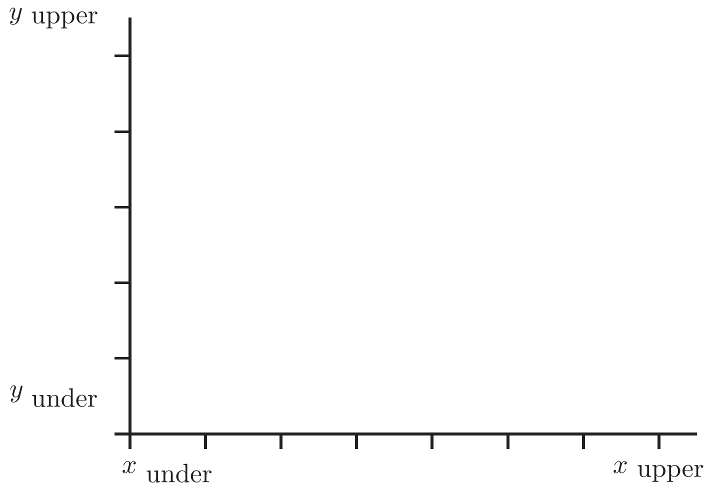


Figure 6.4.1: Definition of x_{under} etc.

number format = (n_x, m_x, n_y, m_y) defines the number of digits of the numbers to be printed along the axis, where nx, ny define the number of digits in front of the decimal point (zero means floating format) and mx, my the number of digits behind the decimal point.
Default: if scales is given (0,2,0,2) otherwise computed by the program.

steps = (stepx, stepy) defines the number of steps to be used along the axis. (default: (10,10))

factor = f defines a multiplication factor. In the case of PLOT VECTOR it defines the multiplication factor of each vector before plotting.
In the case of a function plot, the function is multiplied by f.
Default f=1 in the case of a function plot and automatically scaling in the case of a vector plot. If factor = 0 (default value), this factor is automatically computed, otherwise the length of each vector is multiplied by f before plotting. For the length of the vectors, the physical units are used, where the unit length is made equal to the geometrical unit length as indicated by the co-ordinates.

pict = i of n May be used in combination with the records PLOT FUNCTION, PLOT VELOCITY PROFILE, or TIME HISTORY PLOT. If this statement is used, more than one one-dimensional plot is made in one picture with axes. Statements of this type must be placed consecutively, without other type of statements between. The number i must be given in increasing order from 1 to n. n gives the number of curves to be plotted in one picture.

For example the syntax in the case of $n = 3$ should be:

```
PLOT FUNCTION V%name_1, ... , pict 1 of 3
PLOT FUNCTION V%name_2, ... , pict 2 of 3
PLOT FUNCTION V%name_3, ... , pict 3 of 3
```

angle = α This parameter gives the angle under which the observer sees the plot.
$0 \leq \alpha \leq 360$

The plot parameters defined in a plot record are only valid for that specific plot record. They overwrite defaults locally. Parameters defined by the DEFINE plot parameters command are used for all records.

Colour table

The numbers in the colour table define the colours to be used for the plotting. Which colours are connected with these numbers depends on your local installation.
The default colour table is defined by the numbers 1, 2, 3,...
By the command DEFINE COLOUR TABLE = $(C_1, C_2, C_3, \dots)$ the user may connect new numbers to the colours 1, 2, 3 etc.

6.5 Special commands for time-dependent problems with respect to program SEPPOST”

The general input for the program SEPPOST is described in 6.2. This paragraph is devoted to the available time commands.

The syntax of the time commands is:
Options are indicated between the square brackets [and].

```
TIME = t0
TIME = (t0, t1)
TIME = (t0, t1, istep)
TIME HISTORY [(t0, t1)] print min V%name
TIME HISTORY [(t0, t1)] print max V%name
TIME HISTORY [(t0, t1)] print min abs (V%name)
TIME HISTORY [(t0, t1)] print max abs (V%name)
TIME HISTORY [(t0, t1)] print point(x,y,z) V%name [degfd=k]
TIME HISTORY [(t0, t1)] plot min V%name
TIME HISTORY [(t0, t1)] plot max V%name
TIME HISTORY [(t0, t1)] plot min abs (V%name)
TIME HISTORY [(t0, t1)] plot max abs (V%name)
TIME HISTORY [(t0, t1)] plot point(x,y,z) V%name [degfd=k]
```

TIME = (t_0, t_1, istep) is meant for time-dependent problems. All commands after this COMMAND are carried out for the actual times t_0 to t_1 with integer steps istep. If t_0 and /or t_1 do not coincide with times at which the solution is actually computed, the times closest to t_0 and t_1 are chosen. If t_1 is omitted only $t = t_0$ is used. istep gives the number of time steps minus one between succeeding times (default 1).

TIME HISTORY (t_0, t_1) makes a time history of the quantity from time t_0 to t_1. If (t_0, t_1) is omitted, the complete time interval is used.

plot / print min/max V%name plots or prints the minimum, maximum value of V%name respectively, abs(V%name) does the same for the absolute value of V%name.

plot / print point (x,y,z) V%name makes a time history of the value of V%name in point (x,y,z). At this moment the node closest to (x,y,z) is used instead of the point itself.

7 Some examples of complete SEPRAN runs

7.1 A potential problem in a L-shaped region

As an example we consider the solution of a potential problem in a L-shaped region, consisting of two regions S_1 and S_2 with different permeability constants $\mu(S_1)$ and $\mu(S_2)$. At the upper boundary C_5 the potential is equal to 1, at the lower boundary C_1 the potential is equal to 0. The other outer boundaries may be considered as insulators. The fluxes at the intersection of the region S_1 to S_2 must be continuous.
For a definition of the region as well as its corresponding geometrical quantities, see Figure 7.1.1

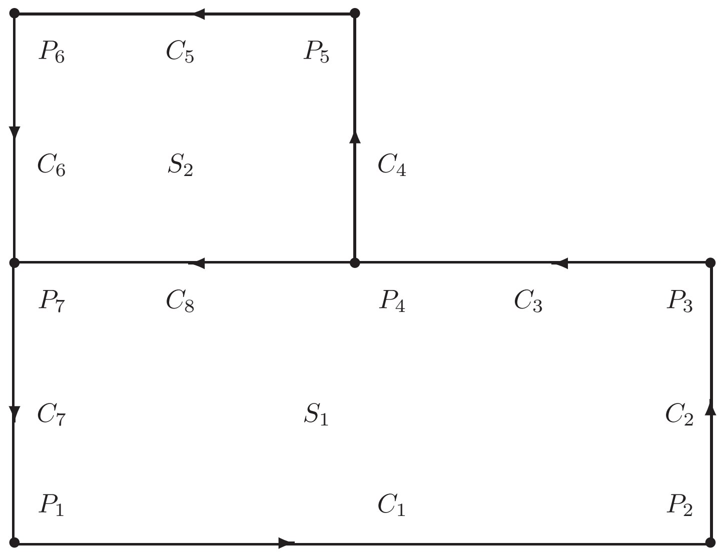


Figure 7.1.1: Definition of the L-shaped region with corresponding geometrical quantities

The mathematical formulation of this problem may be described as follows:
The potential problem is defined by

$$-\text{ div } \mu \bigtriangledown \phi = 0$$

with

$$\mu(S_1) = 1, \mu(S_2) = 2.$$

The boundary conditions are given by:

$$\phi(C_1) = 0, \quad \phi(C_5) = 1$$

$\mu \frac{\partial \phi}{\partial n} = 0$ along the curves C_2, C_3, C_4, C_6 and C_7.

The interface condition at boundary C_8 is given by:

$$\mu(S_1)\frac{\partial \phi}{\partial n}(S_1) = -\mu(S_2)\frac{\partial \phi}{\partial n}(S_2)$$

n denotes the outward normal
The boundary condition $\mu \frac{\partial \phi}{\partial n} = 0$ is a so-called natural boundary condition requiring no special arrangements in the finite element method. The same is the case for the coupling condition at C_8. So in fact these boundary conditions are not given explicitly, but by not prescribing anything they are satisfied automatically.

The easiest way to define the two values of μ in the regions S_1 and S_2 is to define two different element groups. So each element group is connected to a different value of the permeability.

The student must create 4 files in this particular case:

```
practicum7-1.msh
practicum7-1.f
practicum7-1.prb
practicum7-1.pst
```

The file `practicum7-1.msh` contains the mesh input.
The file `practicum7-1.f` contains the main program.
Use the command `sepgetpract practicum7-1` to get a text file containing a part of the Fortran file into your local directory.
The file `practicum7-1.prb` contains the input for the computational program.
The file `practicum7-1.pst` contains the input for SEPPOST.

For the creation of the mesh the definition of Figure 7.1.1 is followed exactly. For both surfaces the surface generator general is used.

The commands to be carried out are:

```
sepmesh practicum7-1.msh
seplink practicum7-1
practicum7-1 < practicum7-1.prb
seppost practicum7-1.pst
```

Mark that the next command may only be carried out if you are sure that the previous one has been finished successfully.

There is no need to retype these files yourself.
You can copy them into your local directory by the command:

```
sepgetex practicum7-1
```

The mesh is created by SEPMESH with the following input file:

```
*******************************************************************************
*
*      File:  practicum7-1.msh
*
*      Contents:  Input for mesh generation part of the example as described
*                 in the SEPRAN Introduction 7.1
*
*      Usage:    sepmesh practicum7-1.msh
*
*******************************************************************************
*
*
*
*
constants                            # Constants so that everything can be
                                     # changed easily
                                     # See Section 4.3
   reals
      x_left  = 0                    # x-coordinate of left-hand side
      y_under = 0                    # y-coordinate of lower side
      x_right = 2                     # x-coordinate of right-hand side
      y_upper = 2                     # y-coordinate of upper side
      x_mid   = 1                     # x-coordinate of C4
      y_mid   = 1                     # y-coordinate of C4
   integers
      nelmlow = 10                    # Number of elements along lower side
      nelmv_1 =  5                    # Number of elements along C2 and C7
      nelmv_2 = 10                    # Number of elements along C4 and C6
      nelmh_1 = 10                    # Number of elements along C3
      nelmupp =  5                    # Number of elements along upper side
      nelmmid = 10                    # Number of elements along C8
end
mesh2d                                # two-dimensional problem
                                      # See Section 4.4
   points                             # Define coordinates of user points
      p1=($x_left ,$y_under)
      p2=($x_right,$y_under)
      p3=($x_right,$y_mid  )
      p4=($x_mid  ,$y_mid  )
      p5=($x_mid  ,$y_upper)
      p6=($x_left ,$y_upper)
      p7=($x_left ,$y_mid  )
   curves                                # Define all the curves
      c1 = line(p1,p2,nelm=$nelmlow)
      c2 = line(p2,p3,nelm=$nelmv_1)
      c3 = line(p3,p4,nelm=$nelmh_1)
      c4 = line(p4,p5,nelm=$nelmv_2)
      c5 = line(p5,p6,nelm=$nelmupp)
      c6 = line(p6,p7,nelm=$nelmv_2)
      c7 = line(p7,p1,nelm=$nelmv_1)
      c8 = line(p4,p7,nelm=$nelmmid)
   surfaces                              # Define all the surfaces
      s1 = general3 (c1,c2,c3,c8,c7)
      s2 = general3 (-c8,c4,c5,c6)
   meshsurf                              # Both surfaces get a separate element
                                         # group number
      selm1 = s1
      selm2 = s2
   plot
end
```

Figure 7.1.2 shows the mesh created by SEPMESH.

Once the mesh has been generated, it is necessary to run the computational program. For the numerical analysis lab it is obligatory to write your own element subroutine. Hence your own main program is also required.
Following the Lecture notes *Numerieke methoden voor partiele differentiaalvergelijkingen* the element matrix is defined by:

$$e^{11} = x_2^2 - x_2^3, e^{21} = x_2^3 - x_2^1, e^{31} = x_2^1 - x_2^2 \tag{7.1.2}$$

$$e^{12} = x_1^3 - x_1^2, e^{22} = x_1^1 - x_1^3, e^{32} = x_1^2 - x_1^1 \tag{7.1.3}$$

$$\Delta = (x_1^2 - x_1^1)(x_2^3 - x_2^2) - (x_2^2 - x_2^1)(x_1^3 - x_1^2) \tag{7.1.4}$$

$$\nabla\phi_i = \begin{pmatrix} \frac{e^{i1}}{\Delta} \\ \frac{e^{i2}}{\Delta} \end{pmatrix} \tag{7.1.5}$$

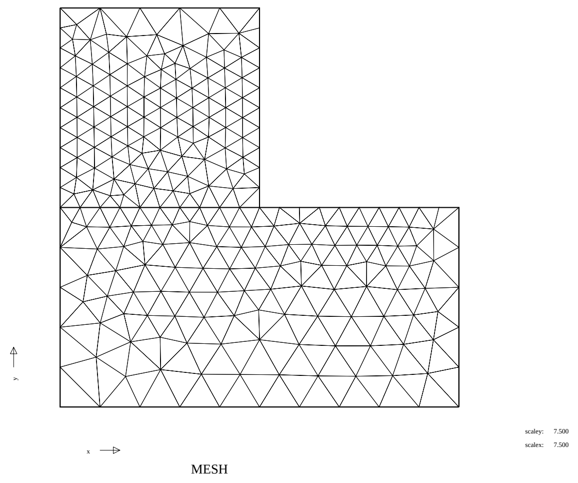


Figure 7.1.2: Mesh plot of L-shaped region

$$S_{ij} = |\Delta|(\mu \nabla \phi_i \cdot \nabla \phi_j) \tag{7.1.6}$$

The following main program and element subroutine may be used for the solution of the potential problem:

```
      program practicum7_1

c     --- Sample program for the example in the introduction Section 7.1

      call sepcom ( 0 )
      end

c     --- It is necessary to define your own element subroutine

      subroutine elemsubr ( ndim, npelm, x, nunk_pel, elem_mat,
     +                      elem_vec, elem_mass, uold, itype )
c ======================================================================
c
c        programmer     Guus Segal
c        version 1.0    date 15-10-1998
c
c
c   copyright (c) 1998-1998  "Ingenieursbureau SEPRA"
c   permission to copy or distribute this software or documentation
c   in hard copy or soft copy granted only by written license
c   obtained from "Ingenieursbureau SEPRA".
c   all rights reserved. no part of this publication may be reproduced,
c   stored in a retrieval system ( e.g., in memory, disk, or core)
c   or be transmitted by any means, electronic, mechanical, photocopy,
c   recording, or otherwise, without written permission from the
c   publisher.
c **********************************************************************
c
c                        DESCRIPTION
c
```

```
c     Special user element subroutine to be used by SEPRAN in the case of
c     the Numerical Analysis Lab of Delft University of Technology
c *********************************************************************
c
c                        KEYWORDS
c
c    matrix
c    vector
c    element
c *********************************************************************
c
c                        MODULES USED
c
c *********************************************************************
c
c                        COMMON BLOCKS
c
c *********************************************************************
c
c                        INPUT / OUTPUT PARAMETERS
c
      implicit none
      integer ndim, npelm, nunk_pel, itype
      double precision x(npelm,ndim), elem_mat(nunk_pel,nunk_pel),
     +                 elem_vec(nunk_pel), elem_mass(nunk_pel),
     +                 uold(nunk_pel)

c     elem_mass   o   In this two-dimensional array the student must store the
c                     element mass matrix, provided the mass matrix must be
c                     computed, in the following way:
c                     elem_mass(i,j) = s_{ij} ; i,j = 1(1)nunk_pel
c                     This matrix should only be filled if a mass matrix is
c                     required, for example for time-dependent problems.
c     elem_mat    o   In this two-dimensional array the student must store the
c                     element matrix, in the following way:
c                     elem_mat(i,j) = s_{ij} ; i,j = 1(1)nunk_pel.
c                     The degrees of freedom in an element are stored
c                     sequentially, first all degrees of freedom corresponding
c                     to the first point, then to the second, etcetera.
c     elem_vec    o   In this array the student must store the element vector,
c                     in the following way:
c                     elem_vec(i) = f_i; i = 1(1)nunk_pel
c                     It concerns the derived quantity that must be computed
c     itype       i   Type number of element.
c                     This parameter is defined in the input block PROBLEM
c     ndim        i   Dimension of the space.
c     npelm       i   Number of points per element
c     nunk_pel    i   number of degrees of freedom in the element
c     uold        i   In this array the old solution, as indicated by V1,
c                     is stored. This solution may contain the boundary
c                     conditions only, if the array has been created by
c                     prescribe_boundary_conditions, but also a starting vector
c                     if V1 has been created by create or even the previous
c                     solution in an iteration process if nonlinear_equations
c                     is used.
```

```
c     x           i   Contains the coordinates of the nodes of the element using
c                     the local node numbering
c                     x(i,1) contains the x-coordinate of the i-th node in the
c                     element and x(i,2) the y-coordinate of this node
c **********************************************************************
c
c                      LOCAL PARAMETERS
c
      double precision mu, e(3,2), delta, gradphi(3,2)
      integer i, j

c     delta       Jacobian delta of the element
c     e           Contains the factors e^ij according to
c                 e(i,j) = e^ij, i=1,2,3; j=1,2
c     gradphi     Contains the gradient of the basis function phi_i according to
c                 gradphi(i,j) = dphi_i/dx_j, i=1,2,3; j=1,2
c     i           General loop variable
c     j           General loop variable
c     mu          Parameter mu in the differential equation
c **********************************************************************
c
c                      SUBROUTINES CALLED
c
c    none
c **********************************************************************
c
c                      I/O
c
c    none
c **********************************************************************
c
c                      ERROR MESSAGES
c
c    none
c **********************************************************************
c
c                      PSEUDO CODE
c
c    The element matrix, element right-hand side and if the problem so
c    requires the element mass matrix are filled by the user, depending on
c    the parameter itype
c
c    The element matrix and element vector are defined in the Lecture Notes
c    "Numerieke methoden voor partiele differentiaalvergelijkingen"
c    See also the description in the manual for the formulas
c **********************************************************************
c
c                      DATA STATEMENTS
c
c ======================================================================
c
      if ( itype.eq.1 ) then

c     --- Type = 1: mu = 1
```

```
            mu = 1d0

         else

c        --- Type = 2: mu = 2

            mu = 2d0

         end if

c        --- Compute the factors e_ij and delta as defined in the Lecture Notes
c            (pages 97,98)

         e(1,1) = x(2,2) - x(3,2)
         e(2,1) = x(3,2) - x(1,2)
         e(3,1) = x(1,2) - x(2,2)

         e(1,2) = x(3,1) - x(2,1)
         e(2,2) = x(1,1) - x(3,1)
         e(3,2) = x(2,1) - x(1,1)

         delta = e(3,1) * e(1,2) - e(3,2) * e(1,1)

c        --- Compute the gradient of the basis functions as defined in the
c            Lecture Notes

         do j = 1, 2
            do i = 1, 3
               gradphi(i,j) = e(i,j) / delta
            end do
         end do

c        --- Fill the element matrix as defined in the Lecture Notes

         do j = 1, 3
            do i = 1, 3
               elem_mat(i,j) = mu * 0.5d0 * abs(delta) *
     +            ( gradphi(i,1)*gradphi(j,1) + gradphi(i,2)*gradphi(j,2) )
            end do
         end do

c        --- The element vector is zero

         do i = 1, 3
            elem_vec(i) = 0d0
         end do

         end
```

The corresponding input file is:

```
******************************************************************************
*
*      File:  practicum7-1.prb
*
```

```
*      Contents:  Input for computational part of the example as described
*                 in the SEPRAN Introduction 7.1
*
*      Usage:    practicum7-1 < practicum7-1.prb
*
*      It has been supposed that the following actions have been carried out
*      with success:
*
*      sepmesh practicum7-1.msh
*      seplink practicum7-1
******************************************************************************
*
*
constants                 # See Section 4.3
   vector_names
      1: potential
end
*
*  Problem definition, see Section 5.4.1
*
problem
   types                                # Define type numbers per element group
      elgrp1 = (type=1)                 # Element group 1: itype = 1
      elgrp2 = (type=2)                 # Element group 1: itype = 2
   essboundcond                         # Define where essential boundary
                                        # conditions are defined (not there
                                        # value)
      curves(c1)                        # The potential on c1 is given
      curves(c5)                        # The potential on c5 is given
end
*
*  Define the structure of the large matrix and implicitly the type
*  of linear solver, see Section 5.4.2
*
matrix
   method = 1                           # The matrix is symmetrical
                                        # A direct solver will be used
end
*
*  Define non-zero essential boundary conditions, see Section 5.4.3
*
essential boundary conditions, sequence_number = 1
   curves(c5), value=1                  # On c5: phi = 1
                                        # On c1: phi = 0
end
*
*  Information for the linear solver, see Section 5.4.5
*
solve, sequence_number = 1
   positive definite                    # The matrix is positive definite
end
*
*  Information for the output, see Section 5.4.7
*
output, sequence_number = 1             # These statements are superfluous
```

```
end
*
*  Define the structure of the main program, see Section 5.4.11
*  Since the structure used here is standard for linear problems, this
*  part is superfluous
*  Since there is only one vector the part vector = %potential may also
*  be skipped
*
structure
   prescribe_boundary_conditions, sequence_number = 1, vector = %potential
   solve_linear_system, seq_solve = 1, vector = %potential
   output, sequence_number = 1, vector = %potential
end
end_of_sepran_input
```

Program seppost allows us to print and plot the solution. It requires input from the standard input file.
If, for example, we want to print the solution, make a standard contour plot and a coloured contour plot then the following input file may be used:

```
*******************************************************************************
*
*      File:  practicum7-1.pst
*
*      Contents:  Input for post processing part of the example as described
*                 in the SEPRAN Introduction 7.1
*
*      Usage:    seppost practicum7-1.pst
*
*******************************************************************************
*
*
postprocessing    # See Section 6.2

   print v%potential           # See Section 6.3
   plot identification, text='Example of potential problem', origin=(3,18)
   plot contour v%potential  # See Section 6.4
   plot coloured contour v%potential
end
```

Figure 7.1.3 shows the required contour plot and Figure 7.1.4 the blank and white representation of the coloured contour plot.

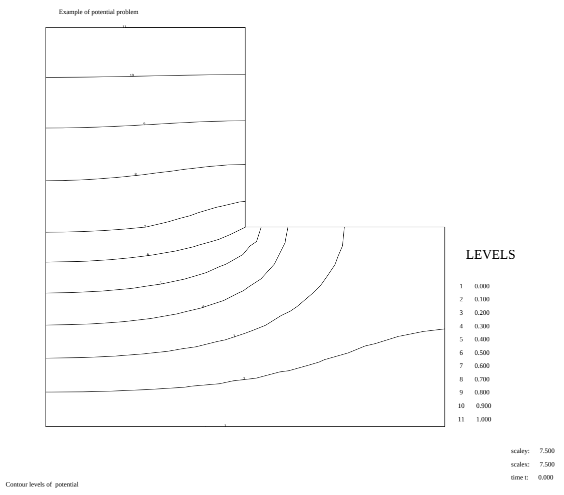


Figure 7.1.3: Contour plot of potential in L-shaped region

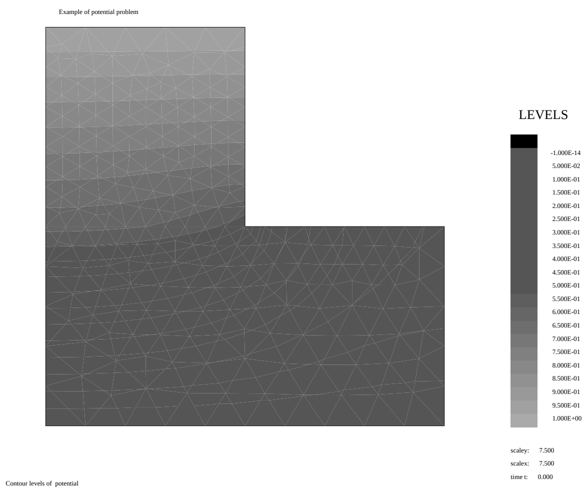


Figure 7.1.4: Black and white representation of coloured contour plot of potential in L-shaped region

7.2 A mathematical test example showing the use of boundary elements

Consider the pure artificial problem:
$\Delta\phi = 0 \ \ x \in (0,1) \times (0,1)$
with boundary conditions:
$\phi = xy$ on curves c1, c2 and c4
$\frac{\partial \phi}{\partial n} = x$ on curve c3.
The region is shown in Figure 7.2.1 The student must create 4 files in this particular case:

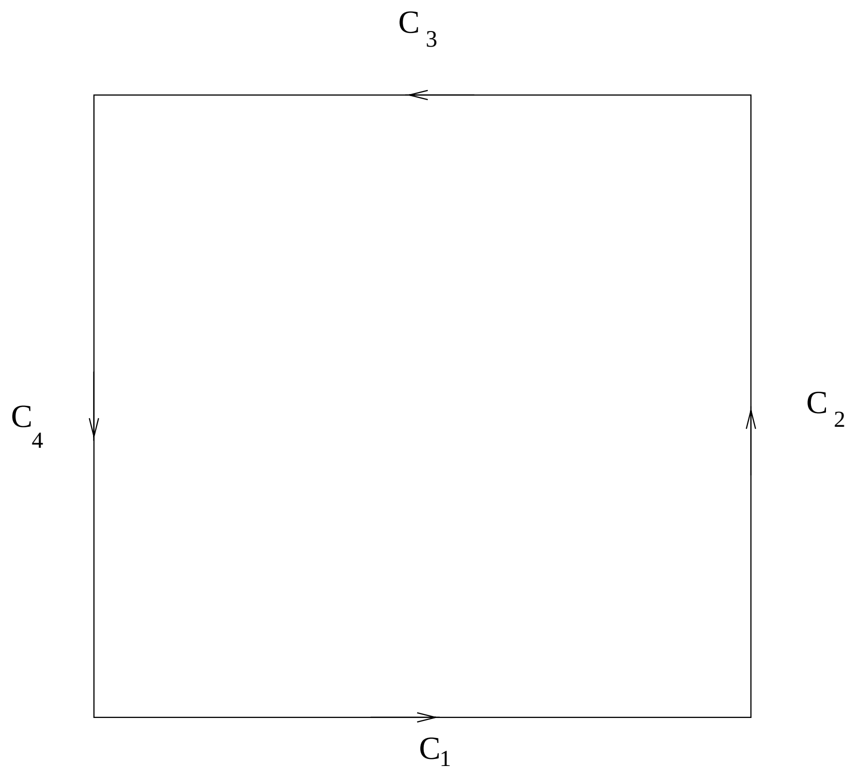


Figure 7.2.1: Region corresponding to artificial test example

```
practicum7-2.msh
practicum7-2.f
practicum7-2.prb
practicum7-2.pst
```

The file `practicum7-2.msh` contains the mesh input.
The file `practicum7-2.f` contains the main program.
Use the command `sepgetpract practicum7-2` to get a text file containing a part of the Fortran file into your local directory.
The file `practicum7-2.prb` contains the input for the computational program.
The file `practicum7-2.pst` contains the input for SEPPOST.

For the creation of the mesh the definition of Figure 7.2.1 is followed exactly. The surface generator quadrilateral is used.

The commands to be carried out are:

```
sepmesh practicum7-2.msh
```

```
seplink practicum7-2
practicum7-2 < practicum7-2.prb
seppost practicum7-2.pst
```

Mark that the next command may only be carried out if you are sure that the previous one has been finished successfully.

In order to get these files into your local directory use the command:

```
sepgetex practicum7-2
```

The mesh is created by SEPMESH with the following input file:

```
*******************************************************************************
*
*      File:  practicum7-2.msh
*
*      Contents:  Input for mesh generation part of the example as described
*                 in the SEPRAN Introduction 7.2
*
*      Usage:    sepmesh practicum7-2.msh
*
*******************************************************************************
*
*
*
*
constants                           # Constants so that everything can be
                                    # changed easily
                                    # See Section 4.3
   reals
      x_left  = 0                   # x-coordinate of left-hand side
      y_under = 0                   # y-coordinate of lower side
      x_right = 1                   # x-coordinate of right-hand side
      y_upper = 1                   # y-coordinate of upper side
   integers
      nelmh = 10                    # Number of elements in horizontal direction
      nelmv = 10                    # Number of elements in vertical direction
end
mesh2d                              # two-dimensional problem
                                    # See Section 4.4
   points                           # Define coordinates of user points
      p1=($x_left ,$y_under)
      p2=($x_right,$y_under)
      p3=($x_right,$y_upper)
      p4=($x_left ,$y_upper)
   curves                              # Define all the curves
      c1 = line(p1,p2,nelm=$nelmh)
      c2 = line(p2,p3,nelm=$nelmv)
      c3 = line(p3,p4,nelm=$nelmh)
      c4 = line(p4,p1,nelm=$nelmv)
   surfaces                            # Define the surface
      s1 = quadrilateral3 (c1,c2,c3,c4)
   plot
end
```

Once the mesh has been generated, it is necessary to run the computational program. For the numerical analysis lab it is obligatory to write your own element subroutine. Hence your own main program is also required.
The element matrix and element vector for the internal element are described in Section 7.1.
With respect to the boundary condition $\frac{\partial\phi}{\partial n} = x$ a boundary element is needed.
The element matrix for this boundary element is zero and the element vector is given by

$$\frac{h}{2}\left[\begin{array}{c} x_1 \\ x_2 \end{array}\right] \tag{7.2.7}$$

The following main program and element subroutine may be used for the solution of the potential problem:

```
      program practicum7_2

c     --- Sample program for the example in the introduction Section 7.2

      call sepcom ( 0 )
      end

c     --- It is necessary to define your own element subroutine

      subroutine elemsubr ( ndim, npelm, x, nunk_pel, elem_mat,
     +                      elem_vec, elem_mass, uold, itype )
c ======================================================================
c
c        programmer     Guus Segal
c        version 1.0    date 15-10-1998
c
c
c   copyright (c) 1998-1998  "Ingenieursbureau SEPRA"
c   permission to copy or distribute this software or documentation
c   in hard copy or soft copy granted only by written license
c   obtained from "Ingenieursbureau SEPRA".
c   all rights reserved. no part of this publication may be reproduced,
c   stored in a retrieval system ( e.g., in memory, disk, or core)
c   or be transmitted by any means, electronic, mechanical, photocopy,
c   recording, or otherwise, without written permission from the
c   publisher.
c **********************************************************************
c
c                        DESCRIPTION
c
c     Special user element subroutine to be used by SEPRAN in the case of
c     the Numerical Analysis Lab of Delft University of Technology
c **********************************************************************
c
c                        KEYWORDS
c
c    matrix
c    vector
c    element
c **********************************************************************
c
```

```
c                        MODULES USED
c
c ********************************************************************
c
c                        COMMON BLOCKS
c
c ********************************************************************
c
c                        INPUT / OUTPUT PARAMETERS
c
      implicit none
      integer ndim, npelm, nunk_pel, itype
      double precision x(npelm,ndim), elem_mat(nunk_pel,nunk_pel),
     +                 elem_vec(nunk_pel), elem_mass(nunk_pel),
     +                 uold(nunk_pel)

c     elem_mass   o   In this two-dimensional array the student must store the
c                     element mass matrix, provided the mass matrix must be
c                     computed, in the following way:
c                     elem_mass(i,j) = s_{ij} ; i,j = 1(1)nunk_pel
c                     This matrix should only be filled if a mass matrix is
c                     required, for example for time-dependent problems.
c     elem_mat    o   In this two-dimensional array the student must store the
c                     element matrix, in the following way:
c                     elem_mat(i,j) = s_{ij} ; i,j = 1(1)nunk_pel.
c                     The degrees of freedom in an element are stored
c                     sequentially, first all degrees of freedom corresponding
c                     to the first point, then to the second, etcetera.
c     elem_vec    o   In this array the student must store the element vector,
c                     in the following way:
c                     elem_vec(i) = f_i; i = 1(1)nunk_pel
c                     It concerns the derived quantity that must be computed
c     itype       i   Type number of element.
c                     This parameter is defined in the input block PROBLEM
c     ndim        i   Dimension of the space.
c     npelm       i   Number of points per element
c     nunk_pel    i   number of degrees of freedom in the element
c     uold        i   In this array the old solution, as indicated by V1,
c                     is stored. This solution may contain the boundary
c                     conditions only, if the array has been created by
c                     prescribe_boundary_conditions, but also a starting vector
c                     if V1 has been created by create or even the previous
c                     solution in an iteration process if nonlinear_equations
c                     is used.
c     x           i   Contains the coordinates of the nodes of the element using
c                     the local node numbering
c                     x(i,1) contains the x-coordinate of the i-th node in the
c                     element and x(i,2) the y-coordinate of this node
c ********************************************************************
c
c                        LOCAL PARAMETERS
c
      double precision e(3,2), delta, gradphi(3,2), h
      integer i, j
```

```
c     delta       Jacobian delta of the element
c     e           Contains the factors e^ij according to
c                 e(i,j) = e^ij, i=1,2,3; j=1,2
c     gradphi     Contains the gradient of the basis function phi_i according to
c                 gradphi(i,j) = dphi_i/dx_j, i=1,2,3; j=1,2
c     h           Length of a boundary element
c     i           General loop variable
c     j           General loop variable
c **********************************************************************
c
c                       SUBROUTINES CALLED
c
c    none
c **********************************************************************
c
c                       I/O
c
c    none
c **********************************************************************
c
c                       ERROR MESSAGES
c
c    none
c **********************************************************************
c
c                       PSEUDO CODE
c
c    The element matrix, element right-hand side and if the problem so
c    requires the element mass matrix are filled by the user, depending on
c    the parameter itype
c
c    The element matrix and element vector are defined in the Lecture Notes
c    "Numerieke methoden voor partiele differentiaalvergelijkingen"
c    See also the description in the manual for the formulas
c **********************************************************************
c
c                       DATA STATEMENTS
c
c ======================================================================
c
      if ( itype.eq.1 ) then

c     --- Type = 1: internal element
c         Compute the factors e_ij and delta as defined in the Lecture Notes
c         (pages 97,98)

         e(1,1) = x(2,2) - x(3,2)
         e(2,1) = x(3,2) - x(1,2)
         e(3,1) = x(1,2) - x(2,2)

         e(1,2) = x(3,1) - x(2,1)
         e(2,2) = x(1,1) - x(3,1)
         e(3,2) = x(2,1) - x(1,1)

         delta = e(3,1) * e(1,2) - e(3,2) * e(1,1)
```

```
c        --- Compute the gradient of the basis functions as defined in the
c            Lecture Notes

         do j = 1, 2
            do i = 1, 3
               gradphi(i,j) = e(i,j) / delta
            end do
         end do

c        --- Fill the element matrix as defined in the Lecture Notes

         do j = 1, 3
            do i = 1, 3
               elem_mat(i,j) = 0.5d0 * abs(delta) *
     +            (gradphi(i,1)*gradphi(j,1)+gradphi(i,2)*gradphi(j,2))
            end do
         end do

c        --- The element vector is zero

         do i = 1, 3
            elem_vec(i) = 0d0
         end do

      else

c     --- Type = 2: boundary element
c         Compute Jacobian h

         h = sqrt ( (x(2,1)-x(1,1))**2 + (x(2,2)-x(1,2))**2 )

c        --- The element matrix is zero

         do j = 1, 2
            do i = 1, 2
               elem_mat(i,j) = 0d0
            end do
         end do

c        --- Fill the element vector

         do i = 1, 2
            elem_vec(i) = h * 0.5d0 * x(i,1)
         end do

      end if

      end

c     --- Since the boundary conditions are space dependent, a extra function
c         subroutine funcbc is necessary

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

c     --- ichoice = 1, phi = xy

         funcbc = x * y

      else

         print *,'Error in funcbc. The value of ichoice is: ', ichoice

      end if

      end
```

The corresponding input file is:

```
*******************************************************************************
*
*      File:  practicum7-2.prb
*
*      Contents:  Input for computational part of the example as described
*                 in the SEPRAN Introduction 7.2
*
*      Usage:    practicum7-2 < practicum7-2.prb
*
*      It has been supposed that the following actions have been carried out
*      with success:
*
*      sepmesh practicum7-2.msh
*      seplink practicum7-2
*******************************************************************************
*
*
constants                 # See Section 4.3
   vector_names
      1: potential
end
*
*  Problem definition, see Section 5.4.1
*
problem
   types                              # Define type numbers per element group
      elgrp1 = (type=1)               # Element group 1: itype = 1
   natbouncond                        # Define types of natural boundary
                                      # conditions
      bngrp1 = (type=2)               # Boundary element group 1: itype = 2
   bounelements                       # Define where natural boundary
                                      # conditions are present
      belm1 = curves(c3)              # Boundary element group 1 is
                                      # defined on c3
   essboundcond                       # Define where essential boundary
                                      # conditions are defined (not there
                                      # value)
      curves(c1 to c2)                  # The potential on c1 to c2 is given
      curves(c4)                        # The potential on c4 is given
end
*
*  Define the structure of the large matrix and implicitly the type
*  of linear solver, see Section 5.4.2
*
matrix
   method = 1                           # The matrix is symmetrical
                                        # A direct solver will be used
end
*
*  Define non-zero essential boundary conditions, see Section 5.4.3
*
essential boundary conditions
   curves(c1 to c2), func=1             # On c1 to c2: phi = xy
   curves(c4), func=1                   # On c4: phi = xy
end
*
*  Information for the linear solver, see Section 5.4.5
*
solve
   positive definite                    # The matrix is positive definite
end
*
*  Information for the output, see Section 5.4.7
*
output                                  # These statements are superfluous
end
*
*  Define the structure of the main program, see Section 5.4.11
*  Since the structure used here is standard for linear problems, this
*  part is superfluous
*
structure
   prescribe_boundary_conditions, sequence_number = 1, vector = %potential
   solve_linear_system, seq_solve = 1, vector = %potential
   output, sequence_number = 1, vector = %potential
end
end_of_sepran_input
```

Program seppost allows us to print and plot the solution. It requires input from the standard input file.
If, for example, we want to print the solution, make a standard contour plot and a coloured contour plot then the following input file may be used:

```
******************************************************************************
*
*      File:  practicum7-2.pst
*
*      Contents:  Input for post processing part of the example as described
*                 in the SEPRAN Introduction 7.2
*
*      Usage:    seppost practicum7-2.pst
```

```
*
******************************************************************************
*
*
postprocessing    # See Section 6.2

   print v%potential           # See Section 6.3
   plot identification, text='Example of potential problem', origin=(3,18)
   plot contour v%potential    # See Section 6.4
   plot coloured contour v%potential
end
```

7.3 An artificial complex example

In this section we demonstrate how complex problems may be solved by SEPRAN. To that end we define the pure artificial problem:
$\Delta\phi = 0 \;\; x \in (0,1) \times (0,1)$
where ϕ is a complex quantity.
The boundary conditions for this problem are:
$\phi = (1,0)$ on curves c1 and c2.
$\phi = (0.1)$ on curve c4.
$\frac{\partial \phi}{\partial n} = 0$ on curve c3.
The region is shown in Figure 7.2.1 The student must create 4 files in this particular case:

```
practicum7-3.msh
practicum7-3.f
practicum7-3.prb
practicum7-3.pst
```

The file `practicum7-3.msh` contains the mesh input.
The file `practicum7-3.f` contains the main program.
Use the command `sepgetpract practicum7-3` to get a text file containing a part of the Fortran file into your local directory.
The file `practicum7-3.prb` contains the input for the computational program.
The file `practicum7-3.pst` contains the input for SEPPOST.

For the creation of the mesh the definition of Figure 7.2.1 is followed exactly. The surface generator quadrilateral is used.

The commands to be carried out are:

```
sepmesh practicum7-3.msh
seplink practicum7-3
practicum7-3 < practicum7-3.prb
seppost practicum7-3.pst
```

Mark that the next command may only be carried out if you are sure that the previous one has been finished successfully.

In order to get these files into your local directory use the command:

```
sepgetex practicum7-3
```

The mesh is created by SEPMESH with the following input file:

```
*****************************************************************************
*
*      File:  practicum7-3.msh
*
*      Contents:  Input for mesh generation part of the example as described
*                 in the SEPRAN Introduction 7.3
*
*      Usage:    sepmesh practicum7-3.msh
*
*****************************************************************************
*
*
*
```

```
*
constants                                   # Constants so that everything can be
                                            # changed easily
                                            # See Section 4.3
   reals
      x_left  = 0                           # x-coordinate of left-hand side
      y_under = 0                           # y-coordinate of lower side
      x_right = 1                           # x-coordinate of right-hand side
      y_upper = 1                           # y-coordinate of upper side
   integers
      nelmh = 10                            # Number of elements in horizontal direction
      nelmv = 10                            # Number of elements in vertical direction
end
mesh2d                                      # two-dimensional problem
                                            # See Section 4.4
   points                                   # Define coordinates of user points
      p1=($x_left ,$y_under)
      p2=($x_right,$y_under)
      p3=($x_right,$y_upper)
      p4=($x_left ,$y_upper)
   curves                                      # Define all the curves
      c1 = line(p1,p2,nelm=$nelmh)
      c2 = line(p2,p3,nelm=$nelmv)
      c3 = line(p3,p4,nelm=$nelmh)
      c4 = line(p4,p1,nelm=$nelmv)
   surfaces                                    # Define the surface
      s1 = quadrilateral3 (c1,c2,c3,c4)
   plot
end
```

Once the mesh has been generated, it is necessary to run the computational program. For the numerical analysis lab it is obligatory to write your own element subroutine. Hence your own main program is also required.
The element matrix and element vector for the internal element are described in Section 7.1.
No boundary elements are needed in this case. The following main program and element subroutine may be used for the solution of the potential problem:

```
      program practicum7_3

c     --- Sample program for the example in the introduction Section 7.3

      call sepcom ( 0 )
      end

c     --- It is necessary to define your own element subroutine

      subroutine elemsubr ( ndim, npelm, x, nunk_pel, elem_mat,
     +                      elem_vec, elem_mass, uold, itype )
c ======================================================================
c
c        programmer     Guus Segal
c        version 1.0    date 15-10-1998
c
c
c   copyright (c) 1998-1998  "Ingenieursbureau SEPRA"
```

```
c   permission to copy or distribute this software or documentation
c   in hard copy or soft copy granted only by written license
c   obtained from "Ingenieursbureau SEPRA".
c   all rights reserved. no part of this publication may be reproduced,
c   stored in a retrieval system ( e.g., in memory, disk, or core)
c   or be transmitted by any means, electronic, mechanical, photocopy,
c   recording, or otherwise, without written permission from the
c   publisher.
c ********************************************************************
c
c                       DESCRIPTION
c
c     Special user element subroutine to be used by SEPRAN in the case of
c     the Numerical Analysis Lab of Delft University of Technology
c ********************************************************************
c
c                       KEYWORDS
c
c    matrix
c    vector
c    element
c ********************************************************************
c
c                       MODULES USED
c
c ********************************************************************
c
c                       COMMON BLOCKS
c
c ********************************************************************
c
c                       INPUT / OUTPUT PARAMETERS
c
      implicit none
      integer ndim, npelm, nunk_pel, itype
      double precision x(npelm,ndim), elem_mass(nunk_pel)
      complex * 16 elem_mat(nunk_pel,nunk_pel), elem_vec(nunk_pel),
     +                 uold(nunk_pel)

c     elem_mass   o   In this two-dimensional array the student must store the
c                     element mass matrix, provided the mass matrix must be
c                     computed, in the following way:
c                     elem_mass(i,j) = s_{ij} ; i,j = 1(1)nunk_pel
c                     This matrix should only be filled if a mass matrix is
c                     required, for example for time-dependent problems.
c     elem_mat    o   In this two-dimensional array the student must store the
c                     element matrix, in the following way:
c                     elem_mat(i,j) = s_{ij} ; i,j = 1(1)nunk_pel.
c                     The degrees of freedom in an element are stored
c                     sequentially, first all degrees of freedom corresponding
c                     to the first point, then to the second, etcetera.
c     elem_vec    o   In this array the student must store the element vector,
c                     in the following way:
c                     elem_vec(i) = f_i; i = 1(1)nunk_pel
c                     It concerns the derived quantity that must be computed
```

```
c     itype       i   Type number of element.
c                     This parameter is defined in the input block PROBLEM
c     ndim        i   Dimension of the space.
c     npelm       i   Number of points per element
c     nunk_pel    i   number of degrees of freedom in the element
c     uold        i   In this array the old solution, as indicated by V1,
c                     is stored. This solution may contain the boundary
c                     conditions only, if the array has been created by
c                     prescribe_boundary_conditions, but also a starting vector
c                     if V1 has been created by create or even the previous
c                     solution in an iteration process if nonlinear_equations
c                     is used.
c     x           i   Contains the coordinates of the nodes of the element using
c                     the local node numbering
c                     x(i,1) contains the x-coordinate of the i-th node in the
c                     element and x(i,2) the y-coordinate of this node
c ********************************************************************
c
c                       LOCAL PARAMETERS
c
      double precision e(3,2), delta, gradphi(3,2)
      integer i, j

c     delta       Jacobian delta of the element
c     e           Contains the factors e^ij according to
c                 e(i,j) = e^ij, i=1,2,3; j=1,2
c     gradphi     Contains the gradient of the basis function phi_i according to
c                 gradphi(i,j) = dphi_i/dx_j, i=1,2,3; j=1,2
c     i           General loop variable
c     j           General loop variable
c ********************************************************************
c
c                       SUBROUTINES CALLED
c
c    none
c ********************************************************************
c
c                       I/O
c
c    none
c ********************************************************************
c
c                       ERROR MESSAGES
c
c    none
c ********************************************************************
c
c                       PSEUDO CODE
c
c    The element matrix, element right-hand side and if the problem so
c    requires the element mass matrix are filled by the user, depending on
c    the parameter itype
c
c    The element matrix and element vector are defined in the Lecture Notes
c    "Numerieke methoden voor partiele differentiaalvergelijkingen"
```

```
c    See also the description in the manual for the formulas
c ***********************************************************************
c
c                       DATA STATEMENTS
c
c =======================================================================
c
      if ( itype.eq.1 ) then

c     --- Type = 1: internal element
c         Compute the factors e_ij and delta as defined in the Lecture Notes
c         (pages 97,98)

         e(1,1) = x(2,2) - x(3,2)
         e(2,1) = x(3,2) - x(1,2)
         e(3,1) = x(1,2) - x(2,2)

         e(1,2) = x(3,1) - x(2,1)
         e(2,2) = x(1,1) - x(3,1)
         e(3,2) = x(2,1) - x(1,1)

         delta = e(3,1) * e(1,2) - e(3,2) * e(1,1)

c        --- Compute the gradient of the basis functions as defined in the
c            Lecture Notes

         do j = 1, 2
            do i = 1, 3
               gradphi(i,j) = e(i,j) / delta
            end do
         end do

c        --- Fill the element matrix as defined in the Lecture Notes

         do j = 1, 3
            do i = 1, 3
               elem_mat(i,j) = 0.5d0 * abs(delta) *
     +            (gradphi(i,1)*gradphi(j,1)+gradphi(i,2)*gradphi(j,2))
            end do
         end do

c        --- The element vector is zero

         do i = 1, 3
            elem_vec(i) = 0d0
         end do

      end if

      end
```

The corresponding input file is:

```
****************************************************************************
*
```

```
*      File:  practicum7-3.prb
*
*      Contents:  Input for computational part of the example as described
*                 in the SEPRAN Introduction 7.3
*
*      Usage:    practicum7-3 < practicum7-3.prb
*
*      It has been supposed that the following actions have been carried out
*      with success:
*
*      sepmesh practicum7-3.msh
*      seplink practicum7-3
******************************************************************************
*
*
constants                 # See Section 4.3
   vector_names
      1: potential
end
*
*  Problem definition, see Section 5.4.1
*
problem
   types                                # Define type numbers per element group
      elgrp1 = (type=1)                 # Element group 1: itype = 1
   essboundcond                         # Define where essential boundary
                                        # conditions are defined (not there
                                        # value)
      curves(c1 to c2)                  # The potential on c1 to c2 is given
      curves(c4)                        # The potential on c4 is given
end
*
*  Define the structure of the large matrix and implicitly the type
*  of linear solver, see Section 5.4.2
*
matrix
   method = 3                           # The matrix is symmetrical complex
                                        # A direct solver will be used
end
*
*  Define non-zero essential boundary conditions, see Section 5.4.3
*
essential complex boundary conditions
   curves(c1 to c2), value=(1,0)        # On c1 to c2: phi = (1,0)
   curves(c4), value=(0,1)              # On c4: phi = (0,1)
end
*
*  Information for the linear solver, see Section 5.4.5
*
solve                                  # These statements are superfluous
end
*
*  Information for the output, see Section 5.4.7
*
output                                  # These statements are superfluous
```

```
end
*
*  Define the structure of the main program, see Section 5.4.11
*  Since the structure used here is standard for linear problems, this
*  part is superfluous
*
structure
   prescribe_boundary_conditions, sequence_number = 1, vector = %potential
   solve_linear_system, seq_solve = 1, vector = %potential
   output, sequence_number = 1, vector = %potential
end
end_of_sepran_input
```

Program seppost allows us to print and plot the solution. It requires input from the standard input file.
If, for example, we want to print the solution, make a standard contour plot and a coloured contour plot then the following input file may be used:

```
******************************************************************************
*
*       File:  practicum7-3.pst
*
*       Contents:  Input for post processing part of the example as described
*                  in the SEPRAN Introduction 7.3
*
*       Usage:    seppost practicum7-3.pst
*
******************************************************************************
*
*
postprocessing    # See Section 6.2

   print v%potential           # See Section 6.3
   plot identification, text='Example of potential problem', origin=(3,18)
   plot contour v%potential    # See Section 6.4
   plot coloured contour v%potential
end
```

7.4 A non-linear potential problem

As an example we consider the solution of a non-linear potential problem in a unit square. So actually we are using the same region as in Section 7.2.

In this special case we want to solve the non-linear potential problem:

$$-\text{ div } \mu \bigtriangledown \phi = e^{-\phi} \tag{7.4.8}$$

with $\phi = 0$ on the whole boundary. Since the right-hand side depends on the solution we have to apply a non-linear iteration.
The following Picard type iteration could be applied:

```
$\phi^0 = 0$
$\varepsilon = 10^{-3}$
Diff $= 1$
$k = 1$
**while** Diff $> \varepsilon$ **do**
  Solve: $-$ div $\mu \bigtriangledown \phi^k = e^{-\phi^{k-1}}$
  Diff $= ||\phi^k - \phi^{k-1}||$
  $k = k + 1$
**end while**
```

This iteration itself is one of the standard options of SEPRAN and does not have to be programmed by the student. However, in each step of the iteration the linear partial differential equation$-$ div $\mu \bigtriangledown \phi^k = e^{-\phi^{k-1}}$ must be solved. This equation requires the building of a matrix and right-hand side and hence a corresponding element subroutine.
The element matrix in this case is of the same shape as in Section 7.1. The only difference is the element vector, which is non zero, due to the presence of a non-vanishing right-hand side.
Following the Lecture notes *Numerieke methoden voor partiele differentiaalvergelijkingen* the element vector is given by

$$f_i^e = \frac{|\Delta|}{6} \phi_i^{k-1}, \tag{7.4.9}$$

provided a Newton Cotes integration rule is applied.

The student must create 4 files in this particular case:

```
practicum7-4.msh
practicum7-4.f
practicum7-4.prb
practicum7-4.pst
```

The file `practicum7-4.msh` contains the mesh input.
The file `practicum7-4.f` contains the main program.
Use the command `sepgetpract practicum7-4` to get a text file containing a part of the Fortran file into your local directory.
The file `practicum7-4.prb` contains the input for the computational program.
The file `practicum7-4.pst` contains the input for SEPPOST.

The commands to be carried out are:

```
sepmesh practicum7-4.msh
seplink practicum7-4
practicum7-4 < practicum7-4.prb
seppost practicum7-4.pst
```

Mark that the next command may only be carried out if you are sure that the previous one has been finished successfully.

The mesh is created by SEPMESH with the following input file:

```
*******************************************************************************
*
*      File:  practicum7-4.msh
*
*      Contents:  Input for mesh generation part of the example as described
*                 in the SEPRAN Introduction 7.4
*
*      Usage:    sepmesh practicum7-4.msh
*
*******************************************************************************
*
*
*
*
constants                           # Constants so that everything can be
                                    # changed easily
                                    # See Section 4.3
   reals
      x_left  = 0                   # x-coordinate of left-hand side
      y_under = 0                   # y-coordinate of lower side
      x_right = 1                   # x-coordinate of right-hand side
      y_upper = 1                   # y-coordinate of upper side
   integers
      nelmh = 10                    # Number of elements in horizontal direction
      nelmv = 10                    # Number of elements in vertical direction
end
mesh2d                              # two-dimensional problem
                                    # See Section 4.4
   points                           # Define coordinates of user points
      p1=($x_left ,$y_under)
      p2=($x_right,$y_under)
      p3=($x_right,$y_upper)
      p4=($x_left ,$y_upper)
   curves                              # Define all the curves
      c1 = line(p1,p2,nelm=$nelmh)
      c2 = line(p2,p3,nelm=$nelmv)
      c3 = line(p3,p4,nelm=$nelmh)
      c4 = line(p4,p1,nelm=$nelmv)
   surfaces                            # Define the surface
      s1 = quadrilateral3 (c1,c2,c3,c4)
   plot
end
```

Once the mesh has been generated, it is necessary to run the computational program.
The following main program and element subroutine may be used for the solution of the non-linear potential problem:

```
      program practicum7_4

c     --- Sample program for the example in the introduction Section 7.4

      call sepcom ( 0 )
      end

c     --- It is necessary to define your own element subroutine
```

```
      subroutine elemsubr ( ndim, npelm, x, nunk_pel, elem_mat,
     +                      elem_vec, elem_mass, uold, itype )
c ======================================================================
c
c        programmer     Guus Segal
c        version 1.0    date 15-10-1998
c
c
c   copyright (c) 1998-1998  "Ingenieursbureau SEPRA"
c   permission to copy or distribute this software or documentation
c   in hard copy or soft copy granted only by written license
c   obtained from "Ingenieursbureau SEPRA".
c   all rights reserved. no part of this publication may be reproduced,
c   stored in a retrieval system ( e.g., in memory, disk, or core)
c   or be transmitted by any means, electronic, mechanical, photocopy,
c   recording, or otherwise, without written permission from the
c   publisher.
c **********************************************************************
c
c                      DESCRIPTION
c
c     Special user element subroutine to be used by SEPRAN in the case of
c     the Numerical Analysis Lab of Delft University of Technology
c **********************************************************************
c
c                      KEYWORDS
c
c    matrix
c    vector
c    element
c **********************************************************************
c
c                      MODULES USED
c
c **********************************************************************
c
c                      COMMON BLOCKS
c
c **********************************************************************
c
c                      INPUT / OUTPUT PARAMETERS
c
      implicit none
      integer ndim, npelm, nunk_pel, itype
      double precision x(npelm,ndim), elem_mat(nunk_pel,nunk_pel),
     +                 elem_vec(nunk_pel), elem_mass(nunk_pel),
     +                 uold(nunk_pel)

c     elem_mass   o   In this two-dimensional array the student must store the
c                     element mass matrix, provided the mass matrix must be
c                     computed, in the following way:
c                     elem_mass(i,j) = s_{ij} ; i,j = 1(1)nunk_pel
c                     This matrix should only be filled if a mass matrix is
c                     required, for example for time-dependent problems.
```

```
c     elem_mat    o   In this two-dimensional array the student must store the
c                     element matrix, in the following way:
c                     elem_mat(i,j) = s_{ij} ; i,j = 1(1)nunk_pel.
c                     The degrees of freedom in an element are stored
c                     sequentially, first all degrees of freedom corresponding
c                     to the first point, then to the second, etcetera.
c     elem_vec    o   In this array the student must store the element vector,
c                     in the following way:
c                     elem_vec(i) = f_i; i = 1(1)nunk_pel
c                     It concerns the derived quantity that must be computed
c     itype       i   Type number of element.
c                     This parameter is defined in the input block PROBLEM
c     ndim        i   Dimension of the space.
c     npelm       i   Number of points per element
c     nunk_pel    i   number of degrees of freedom in the element
c     uold        i   In this array the old solution, as indicated by V1,
c                     is stored. This solution may contain the boundary
c                     conditions only, if the array has been created by
c                     prescribe_boundary_conditions, but also a starting vector
c                     if V1 has been created by create or even the previous
c                     solution in an iteration process if nonlinear_equations
c                     is used.
c     x           i   Contains the coordinates of the nodes of the element using
c                     the local node numbering
c                     x(i,1) contains the x-coordinate of the i-th node in the
c                     element and x(i,2) the y-coordinate of this node
c **********************************************************************
c
c                       LOCAL PARAMETERS
c
      double precision mu, e(3,2), delta, gradphi(3,2)
      integer i, j

c     delta       Jacobian delta of the element
c     e           Contains the factors e^ij according to
c                 e(i,j) = e^ij, i=1,2,3; j=1,2
c     gradphi     Contains the gradient of the basis function phi_i according to
c                 gradphi(i,j) = dphi_i/dx_j, i=1,2,3; j=1,2
c     i           General loop variable
c     j           General loop variable
c     mu          Parameter mu in the differential equation
c **********************************************************************
c
c                       SUBROUTINES CALLED
c
c    none
c **********************************************************************
c
c                       I/O
c
c    none
c **********************************************************************
c
c                       ERROR MESSAGES
c
```

```
c    none
c ********************************************************************
c
c                       PSEUDO CODE
c
c    The element matrix, element right-hand side and if the problem so
c    requires the element mass matrix are filled by the user, depending on
c    the parameter itype
c
c    The element matrix and element vector are defined in the Lecture Notes
c    "Numerieke methoden voor partiele differentiaalvergelijkingen"
c    See also the description in the manual for the formulas
c ********************************************************************
c
c                       DATA STATEMENTS
c
c ====================================================================
c
      if ( itype.eq.1 ) then

c     --- Type = 1: mu = 1

         mu = 1d0

      end if

c     --- Compute the factors e_ij and delta as defined in the Lecture Notes
c          (pages 97,98)

      e(1,1) = x(2,2) - x(3,2)
      e(2,1) = x(3,2) - x(1,2)
      e(3,1) = x(1,2) - x(2,2)

      e(1,2) = x(3,1) - x(2,1)
      e(2,2) = x(1,1) - x(3,1)
      e(3,2) = x(2,1) - x(1,1)

      delta = e(3,1) * e(1,2) - e(3,2) * e(1,1)

c     --- Compute the gradient of the basis functions as defined in the
c         Lecture Notes

      do j = 1, 2
         do i = 1, 3
            gradphi(i,j) = e(i,j) / delta
         end do
      end do

c     --- Fill the element matrix as defined in the Lecture Notes

      do j = 1, 3
         do i = 1, 3
            elem_mat(i,j) = mu * 0.5d0 * abs(delta) *
     +         ( gradphi(i,1)*gradphi(j,1) + gradphi(i,2)*gradphi(j,2) )
         end do
      end do

c     --- The element vector is defined by the previous solution

      do i = 1, 3
         elem_vec(i) = abs(delta)/6d0*exp(-uold(i))
      end do

      end
```

The corresponding input file is:

```
*******************************************************************************
*
*      File:  practicum7-4.prb
*
*      Contents:  Input for computational part of the example as described
*                 in the SEPRAN Introduction 7.4
*
*      Usage:    practicum7-4 < practicum7-4.prb
*
*      It has been supposed that the following actions have been carried out
*      with success:
*
*      sepmesh practicum7-4.msh
*      seplink practicum7-4
*******************************************************************************
*
*
constants                 # See Section 4.3
   vector_names
      1: potential
end
*
*  Problem definition, see Section 5.4.1
*
problem
   types                                # Define type numbers per element group
      elgrp1 = (type=1)                 # Element group 1: itype = 1
   essboundcond                         # Define where essential boundary
                                        # conditions are defined (not there
                                        # value)
      curves(c1 to c4)                  # The potential is given on c1 ... c4
end
*
*  Define the structure of the large matrix and implicitly the type
*  of linear solver, see Section 5.4.2
*
matrix
   method = 1                           # The matrix is symmetrical
                                        # A direct solver will be used
end
*
*  Define non-zero essential boundary conditions, see Section 5.4.3
*
```

```
essential boundary conditions, sequence_number = 1
   value=0                  # The value on the boundary is 0
                            # Since this is the default this statement is
                            # superfluous
end
*
*  Information for the linear solver, see Section 5.4.5
*
solve, sequence_number = 1
   positive definite                    # The matrix is positive definite
end
*
*  Information for the non-linear solver, see Section 5.4.6
*
nonlinear_equations, sequence_number = 1
   global_options, maxiter=10, accuracy=1e-3, print_level=2  # global options
   equation 1                               # there is only one equation
      local_options, iteration_method = standard  # local option, since this
                                                 # is the default, this is
                                                 # superfluous
end
*
*  Information for the output, see Section 5.4.7
*
output, sequence_number = 1             # These statements are superfluous
end
*
*  Define the structure of the main program, see Section 5.4.11
*  Since the structure used here is standard for linear problems, this
*  part is superfluous
*  Since there is only one vector the part vector = %potential may also
*  be skipped
*
structure
   prescribe_boundary_conditions, sequence_number = 1, vector = %potential
   solve_nonlinear_system, sequence_number = 1, vector = %potential
   output, sequence_number = 1, vector = %potential
end
end_of_sepran_input
```

Program seppost allows us to print and plot the solution. It requires input from the standard input file.
If, for example, we want to print the solution, make a standard contour plot and a coloured contour plot then the following input file may be used:

```
******************************************************************************
*
*      File:  practicum7-4.pst
*
*      Contents:  Input for post processing part of the example as described
*                 in the SEPRAN Introduction 7.4
*
*      Usage:    seppost practicum7-4.pst
*
******************************************************************************
```

```
*
*
postprocessing    # See Section 6.2

   print v%potential           # See Section 6.3
   plot identification, text='Example of non-linear potential problem'//
      origin=(3,18)
   plot contour v%potential  # See Section 6.4
   plot coloured contour v%potential
end
```

Index