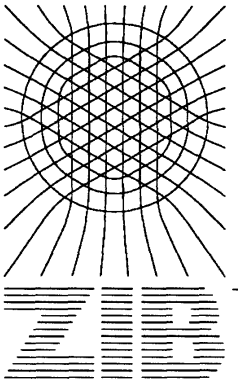
Konrad-Zuse-Zentrum für Informationstechnik Berlin



Rainer Roitzsch

KASKADE User's Manual

(Version 1.0)

Technical Report TR 89–4 (August 1989)

I

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Heilbronner Strasse 10
1000 Berlin 31
Verantwortlich: Dr. Klaus André
Umschlagsatz und Druck: Rabe KG Buch- und Offsetdruck Berlin

ISSN 0933-789X

Contents

1 Introduction **3**

1.1 Mathematical context 3

1.2 Software context 4

2 The command language **5**

2.1 Overview 5

2.2 Commands of "**Command**" 7

2.2.1 `Quit` 7

2.2.2 `Help` [⟨*commandName*⟩] or `?` 7

2.2.3 `Do` ⟨*fileName*⟩ 7

2.2.4 `Time` 7

2.2.5 `CmdPrint` [`Names`][`Settings`] 7

2.2.6 `MsgPrint` 8

2.3 Commands of "**Triangulation**" 8

2.3.1 `ReadTri` ⟨*fileName*⟩ 8

2.3.2 `WriteTri` ⟨*fileName*⟩ [`level` [⟨*levelNo*⟩]] 9

2.3.3 `InfTri` 10

2.3.4 `SelTri` ⟨*triangName*⟩ 10

2.3.5 `DeleteTri` [⟨*triangName*⟩] 11

2.3.6 `CheckTri` 11

2.3.7 `Refine` [`all` | `first last`] 11

2.4 Commands of "**Assembling**" 12

2.4.1 `SelInteg` ⟨*name*⟩ 12

2.4.2 `InfInteg` 12

2.4.3 `Problem` ⟨*name*⟩ 12

2.4.4 `InfProblem` 13

2.4.5 `UpdProblem` ⟨*constName*⟩ ⟨*value*⟩ 13

2.5 Commands of "**Solve**" 13

2.5.1 `SetBreak` ⟨*limitName*⟩ ⟨*value*⟩ 14

2.5.2 `InfSolve` 14

2.5.3 `Solve` [`onestep`] 15

2.5.4 `Direct` 15

2.5.5 `Estimate` 16
2.5.6 `Refine` 16
2.5.7 `Iterate` 16
2.6 Commands of "**Graphic**" 17
2.6.1 `Graphic` [`clear`] [⟨*selectName*⟩] [`clipping` ⟨*rectangle*⟩] [`levels` ⟨*number*⟩] 17
2.6.2 `InfGraphic` 17
2.6.3 `Show` [`PostScript`] [`noframe`] [`continue`] 18
2.6.4 `PostScript` [`size` (⟨*size*⟩)] [`origin` (⟨*point*⟩)] [`font` ⟨*fontName*⟩] [`file` ⟨*fileName*⟩] 18
2.6.5 `InfPS` 18
2.6.6 `Window` [`automatic`] [`top`][`bottom`][`left`] [`right`] [`size` (⟨*size*⟩)] [`origin` (⟨*point*⟩)] [`font` ⟨*fontName*⟩] [`name` ⟨*windowName*⟩] 19
2.6.7 `InfWindow` 19

3 Definition of a new problem **20**

4 An Example **21**

References **23**

1 Introduction

The Users' Manual contains the description of the user interface of the C–KASKADE implementation. A corresponding Pascal implementation is done by P. Leinen [5] at Dortmund. The purpose of this research code is to supply a software environment for the development of adaptive finite element algorithms (2–D). The KASKADE implementation described in this manual is the variant used to solve elliptic partial differential equations [3], other variants are under development (see [4]). Thus efficiency of the code is sometimes sacrificed for clearness or ease of implementation or for generality. Potential users are numerical analysts in need for an environment to test new ideas in the field of adaptive finite element methods.

To understand the internals of the program the user is refered to the Programmer's Manual.

Errors, problem reports, or any comments should be forwarded to the author at the Konrad–Zuse–Zentrum (ZIB), for an e–mail address see the Installation Guide (`README`).

1.1 Mathematical context

The program KASKADE solves elliptic partial differential equations of the type

$$\begin{aligned} -(p_1 u_x)_x - (p_2 u_y)_y + qu &= g \quad \text{in } \Omega \\ u &= \gamma \quad \text{on } \Gamma_0 \subset \partial\Omega \\ \frac{\partial u}{\partial n} &= 0 \quad \text{on } \partial\Omega \backslash \Gamma_0 \ . \end{aligned} \tag{1}$$

p_1, p_2, q, g, and γ are piecewise continuous real valued functions on Ω with $p_1 p_2 > 0$ and $q \geq 0$. $\Omega \subset \mathbb{R}^2$ is a bounded polygonal domain. The program implements the method described by DEUFLHARD, LEINEN, YSERENTANT [3]. KASKADE implements this finite element method for linear elements on triangles.

The user has to define a *coarse* triangulation to represent Ω and Γ_0 (see Section 2.3). The solution on this grid is found by a Cholesky decomposition of the linear system

$$A_i u_i = b_i \tag{2}$$

which results from the weak formulation of (1):

$$f(u) = \tfrac{1}{2} a(u,u) - G(u), \quad u \in H^1_\gamma(\Omega)$$

where

$$a(u,v) = \int_\Omega [p_1 u_x v_x + p_2 u_y v_y + quv] d(x,y)$$

and

$$G(v) = \int_\Omega gv d(x,y).$$

(The notation of [1] is used.)

The solution found by the direct solver is used as the initial level of the cascade and as part of the preconditioner for the conjugate gradient solver on the next cascade levels. (The coarse grid given by the user to define Ω and Γ_0 can be refined uniformly before being used as the initial level.)

These levels contain two phases, first the error estimator is called until a predefined amount of new points is found and second equation (2) is solved by an iterative method.

The error estimator gives information of the error on the edges of triangles by solving a simplified quadratic approximation. If not enough triangles are found to increase the number of points by a given factor (predefined 2.0) the error estimator is called successively. During the refinement process the solution on the new points is interpolated by the solutions at the end points of the refined edge and by the quadratic correction estimated by the error estimator.

The iterative solver (the conjugate gradient method) uses as preconditioner the transformation S to the hierarchical basis and the decomposition from the direct solution

$$S^T D_i^{-1} S$$

where

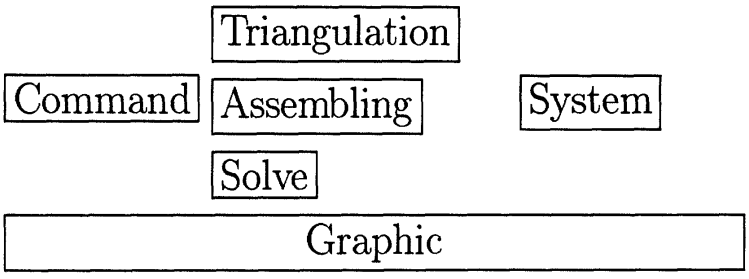
$$D_i := \begin{pmatrix} A_0 & O \\ O & d_i \end{pmatrix}, \quad d_i = \mathrm{diag}(A_i)\,.$$

The iterative solver does not assemble any matrices in an explicit representation, rather matrix vector multiplications are preformed elementwise. The termination criteria for the iterative solver results from the estimated global error multiplied by a safety factor (predefined 1/4). More information on the algorithms implemented can be found in the Programmer's Manual [6].

1.2 Software context

This "batch" version of KASKADE is written in the language C and is developed on a Macintosh II computer using the MPW compiler and environment. No features for interactive graphics or menus are included, therefore "batch" version. It is tested on Sun workstations (Unix).

The program KASKADE consists of the following modules



The command to use these modules are described in Section 2.

KASKADE reads at the start a file `startup.ksk` which contains a sequence of commands and process them. If no `quit` is included, more commands are requested from standard input. This allows batch processing and – to a certain extend – interactive use.

A standard batch run should include commands to

- select the problem,
- read the geometric input data,
- set parameters for the solution method, and
- request output processing.

For further information on the implementation see the Programmer's Manual [6].

2 The command language

2.1 Overview

The KASKADE command language has a simple syntactic structure. A command is delimited by a newline or a command delimiting character. Parameters are separated by white space (e.g. blanks or tabs), exceptions are strings which are quoted by string delimiting characters.

The rest of an input line is ignored after a comment character (predefined '%').

Capital letters are treated as small letters (exceptions are strings).

The basic command language module ("**Command**") contains commands to

- quit the program,
- give some help,
- execute a command file,
- inform on the state of "**Command**" , and
- print time and memory used.

All other commands reflect the modularization of the KASKADE implementation. These modules are

- "**Triangulation**"
  - reading and writing a triangulation,
  - informing on the state of "**Triangulation**" ,
  - setting parameters, and
  - explicit refinement.
- "**Assemble**"
  - choosing parameters for numerical integration (points, weights),
  - informing on the state of "**Assembling**" , and
  - selecting and informing on the problem.
- "**Solve**"
  - setting break conditions for the solution process,
  - informing on the solution process,
  - executing the solution process, and
  - executing the direct solver, error estimator, and iterative solver directly.
- "**Graphic**"
  - setting of the graphic environment,
  - informing on the state of "**Graphic**" ,
  - selecting a driver, and
  - producing graphic output.

The manual now follows this structure. Each command is described as the external interface to its module.

2.2 Commands of "Command"

2.2.1 Quit

Quit stops the program. All files are closed.

2.2.2 Help [⟨*commandName*⟩] or ?

Help prints the list of command names, if used with a parameter, the short description corresponding to ⟨*commandName*⟩ is printed. The list of predefined parameter keywords is printed too.

Example

```
Kaskade: help postscript
Help: postscript : 'set parameters for the postscript driver';
      size       origin     font       file
Kaskade:
```

2.2.3 Do ⟨*fileName*⟩

Do executes the commands in ⟨*fileName*⟩. At the start of KASKADE the file startup.ksk is executed.

Example

```
Do L-shape.ksk
```

2.2.4 Time

Time prints the time and memory used since the start of KASKADE .

Example

```
kaskade:Time
Time: 0.22 secs, since last call 0.03 secs, alloc. mem.: 8KB
kaskade:
```

2.2.5 CmdPrint [Names][Settings]

CmdPrint lists all command names of KASKADE defined at the moment of execution and the actual setting of certain parameters. The names are read from a file during the initialization of "**Command**" . This file has the name kaskcmd.def. The printed information may be selected by the parameters names and settings.

Example

```
kaskade:CmdPrint Settings
Cmd: current settings of the command interpreter
   Prompt:'Kaskade: '
   Maximal number of parameters: 10
   Cases are ignored
   Escape symbol: '\'
   Comment symbol: '%'
kaskade:
```

2.2.6 MsgPrint

`MsgPrint` lists all messages of KASKADE defined at the moment of execution. Messages are read from files during the initialization of the modules. All these files have the postfix `.msg`.

2.3 Commands of "Triangulation"

2.3.1 ReadTri ⟨*fileName*⟩

`ReadTri` reads the geometric input data from the file ⟨*fileName*⟩.

Example

```
kaskade:readtri kschlitz.geo
ReadTri: 6 points, 9 edges, 4 triangles read from kschlitz.geo
kaskade:
```

The file should contain the name of the triangulation in the first line. The triangulation can be referenced by this name through the commands `SelTri` and `DeleteTri` . This line is followed by `Dimension:` and the number of points, edges, triangles, and midpoints of arcs.

For each point the index, the coordinates and a letter to denote the type (`D`: Dirichlet, `N`: Neumann, `I`: interior) should be given.

For each edge the index of the endpoints and the type (`D`, `N`, or `I`) of the edges are needed.

Triangles are defined by their three edges, which should always be mathematically correctly oriented. If edges are arcs the coordinates of the midpoint and the numbers of edges with a list of these edges is followed. Each of these three (four) lists ends with an extra line containing `END`.

The elements of the list of points may be followed by an initial value for the solution.

Example

```
KreisSchlitz
Dimension:(6,9,4,1)
0:(1.0,0.0),D
1:(0.0,1.0),D
2:(-1.0,0.0),D
3:(0.0,-1.0),D
4:(0.0,0.0),D
5:(1.0,0.0),D
END
0:(0,1),D
1:(2,1),D
2:(2,3),D
3:(5,3),D
4:(4,0),D
5:(1,4),I
6:(4,2),I
7:(3,4),I
8:(4,5),D
END
0:(0,5,4)
1:(1,6,5)
2:(2,7,6)
3:(3,8,7)
END
(0,0):4
0
1
2
3
END
```

2.3.2 `WriteTri` ⟨*fileName*⟩ [`level` [⟨*levelNo*⟩]]

`WriteTri` writes the actual triangulation data to the file ⟨*fileName*⟩. This file can be used for future `ReadTri` calls. An older refinement level will be selected by `level` ⟨*levelNo*⟩. If ⟨*levelNo*⟩ is missing after the keyword `level` level 0 is substituted.

Example

```
Kaskade: write newquad.geo
```

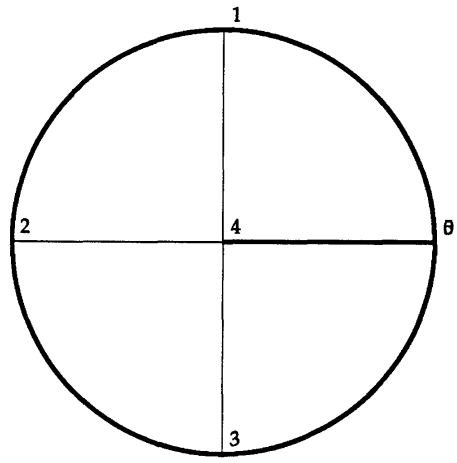


Figure 1: Triangulation of the example

```
WriteTri: 62 lines written to 'newquad.geo' for level 0
Kaskade:
```

2.3.3 InfTri

InfTri lists some information about the current triangulation.

Example

```
kaskade:inftri
Tri: current triangulation 'KreisSchlitz' from 'kschlitz.geo'
       noOfPoints          :    6
       noOfEdges           :    9
       noOfTriangles       :    4
       noOfInitPoints      :    6
       noOfInitEdges       :    9
       noOfInitTriangles   :    4
       refLevel            :    0
       maxDepth            :    0
kaskade:
```

2.3.4 SelTri ⟨*triangName*⟩

SelTri selects a triangulation given by its name ⟨*triangName*⟩ to be the current one.

2.3.5 `DeleteTri` [⟨*triangName*⟩]

`DeleteTri` deletes a triangulation given by its name ⟨*triangName*⟩. The current triangulation is deleted if no parameters are given.

2.3.6 `CheckTri`

`CheckTri` checks the current triangulation for inconsistencies.

2.3.7 `Refine` [`all` | `first last`]

`Refine` refines all triangles, the first or the last one. If no parameters are given the grid is adaptively refined. In this case a call to the error estimator is required.

Example

```
kaskade:refine all
Tri: 9 points, 30 edges, 16 triangles generated
Tri: 0 edges, 0 triangles for green closure
kaskade:
```

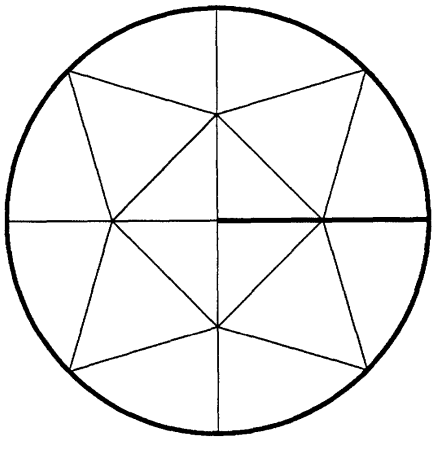


Figure 2: Triangulation after refinement

2.4 Commands of "Assembling"

2.4.1 SelInteg ⟨*name*⟩

SelInteg selects a predefined set of integration points and weights. The following sets are predefined and may be selected by their name: Axelsson from [1], Bank from [2], quadquad, and user.

Example

```
kaskade:selinteg bank
kaskade:infinteg
Assemble: #IP=1, #quadIP=3, #userIP=-1
Assemble: standard integration points
          ( 0.3333, 0.3333)  0.5000
Assemble: standard quadratic integration points
          ( 0.1667, 0.1667)  0.1667
          ( 0.6667, 0.1667)  0.1667
          ( 0.1667, 0.6667)  0.1667
kaskade:
```

2.4.2 InfInteg

InfInteg prints a list of the current setting of integration parameters.

2.4.3 Problem ⟨*name*⟩

Problem is used to select a predefined problem. A problem, i.e. a set of functions p_1, p_2, q, g, γ from (1), may be selected by a name from Table 1.

Std	p_1, p_2, q, g, γ constant
BoundDisc	uncontinuous boundary values
Bank	problem from R.E. Bank [2]
Region	uncontinuous p [3]
Circles	$\Delta u = -4$, $\gamma(x,y) = x^2 + y^2$

Table 1: Predefined problems

Example

```
kaskade:Problem circles
kaskade:InfProblem
```

```
Problem: current problem is 'circles'
            pXConst   =  1.0000e+00
            pYConst   =  1.0000e+00
            qConst    =  0.0000e+00
            gConst    = -4.0000e+00
            BValueConst =  0.0000e+00
kaskade:
```

2.4.4 InfProblem

`InfProblem` lists the name of the current problem and the value of some constants.

2.4.5 UpdProblem ⟨*constName*⟩ ⟨*value*⟩

`UpdProblem` may be used to change some constants, see Table 2.

`pXConst`	p_1 =const
`pYConst`	p_2 =const
`qConst`	q =const
`gConst`	g =const
`BValueConst`	γ =const

Table 2: Constants of `UpdProblem`

Example

```
kaskade:UpdProblem gConst -1.0
kaskade:InfProblem
Problem: current problem is 'circles'
            pXConst   =  1.0000e+00
            pYConst   =  1.0000e+00
            qConst    =  0.0000e+00
            gConst    = -1.0000e+00
            BValueConst =  0.0000e+00
kaskade:
```

2.5 Commands of "Solve"

A solution process always starts with a direct solution of the system defined by the actual triangulation and the given problem. A sequence of calls to the

error estimator and the iterative solver follow until a break condition holds.

2.5.1 SetBreak ⟨*limitName*⟩ ⟨*value*⟩

SetBreak defines certain limits for Solve . The limits with their defaults are in Table 3.

depth	maximum number of refinements of a triangle	10
level	maximum number of refinements	10
dimension	maximum number of unknowns	1000
steps	maximum number of steps	11
globalError	estimated global error	1.0E-4
maxIte	maximum number of iteration steps	100
iteFactor	iteration error factor	0.5
sFactor	factor for number of new points	2.0
sigma	factor for refinement process	0.95

Table 3: Limits of a solution process

The meaning of these limits and parameters are explained in the Programmer's Manual.

2.5.2 InfSolve

InfSolve prints the current limits and the corresponding actual values of the solution process.

Example

```
Kaskade: SetBreak dimension 666
Kaskade: SetBreak level 4
Kaskade: InfSolve
Solve: State of solution process
  level          0 -     4
  depth          0 -    10
  dimension     13 -   666
  global error 1.000e+00 - 1.000e-04
  max steps for Iterate: 100, last iteration error:0.000e+00
  no direct solution available
  no failures yet
Kaskade:
```

2.5.3 Solve [onestep]

Solve continues the solution process until a break condition holds. The output is given in tabular form. The parameter onestep can be used to process one step (direct solver, error estimator, and iterative solver) in the predefined sequence.

Example

```
Kaskade: Solve
  step    level depth  dim    estiErr  iter#    iteErr
Direct        0     0    13
Estimate      1     1    41  4.196e-03
Iterate       1     1    41                1   5.109e-04
Estimate      2     2   145  1.704e-03
Iterate       2     2   145                2   3.829e-04
Estimate      3     3   545  6.717e-04
Iterate       3     3   545                3   1.431e-04
Estimate      4     4  1961  2.013e-04
Iterate       4     4  1961                3   4.244e-05
Kaskade: InfSolve
Solve: State of solution process
  level         4 -     4
  depth         4 -    10
  dimension  1961 -   666
  global error 2.013e-04 - 1.000e-04
  max steps for Iterate: 100, last iteration error:4.244e-05
  direct solution for level 0 available
  no failures yet
Kaskade:
```

2.5.4 Direct

Direct solves the selected problem on the current triangulation using a direct solver. First the global stiffness matrix is assembled, then the resulting linear system is solved (Cholesky).

Example

```
Kaskade: direct
Solve: start of direct cholesky, n=21
Solve: min/max(diag) 2.500e-01/1.000e+00 (ChkSol)
Solve: end of direct cholesky
Kaskade:
```

2.5.5 Estimate

`Estimate` calls the KASKADE error estimator. Values for the adaptive refinement process and an estimate for the global error are computed.

Example

```
Kaskade: Estimate
Solve: estiErr=8.395e-02(2.897460e-01), mbar=6.408e-04
Kaskade:
```

`estiErr` is the estimated global error, the sum of the squares of the residuals in respect to the quadratic solution. The number in parenthesis is the corresponding square root. `mbar` is the weighted sum. For more information see the Programmer's Manual.

2.5.6 Refine

`Refine` (without parameters) invokes the KASKADE adaptive refinement process.

Example

```
Kaskade: Refine
Solve: start adaptive refinement, sigma=0.95, s=2.0, 50 points
Tri:   31 points,  107 edges,   60 triangles generated
Tri:               27 edges,   54 triangles for green closure
Solve: estiErr=4.714e-02(2.171119e-01), mbar=2.024e-04
Tri:  142 points,  533 edges,  332 triangles generated
Tri:               42 edges,   84 triangles for green closure
Solve: end adaptive refinement, 173 new points, 2 level(s)
Kaskade:
```

In the first refinement step not enough new points are found. Therefore the error estimator is called again to find more points. For the meaning of the parameters `sigma` and `s` see the Programmer's Manual.

2.5.7 Iterate

`Iterate` calls the iterative solver (cg).

Example

```
Kaskade: Iterate
Solve: start of CG, n=223, eps=2.642e-03(5.140e-02) (precond.)
Solve: first CG step: res=8.507e-02(2.917e-01)
```

```
Solve: CG step 2: delta=3.234e-02(1.798e-01)
Solve: CG step 3: delta=2.352e-02(1.534e-01)
Solve: CG step 4: delta=8.492e-03(9.215e-02)
Solve: CG step 5: delta=2.327e-03(4.824e-02)
Solve: after max 5 CG steps: res=2.327083e-03
Kaskade:
```

2.6 Commands of "Graphic"

2.6.1 `Graphic` [`clear`] [⟨*selectName*⟩] [`clipping` ⟨*rectangle*⟩] [`levels` ⟨*number*⟩]

`Graphic` is used to set the graphic environment. With the parameters ⟨*selectName*⟩ `boundary`, `solution`, `triangulation`, `index`, `points` parts of the picture which is drawn by the next `Show` are selected. They are added to the current graphics selection. `clear` clears this selection. A parameter `clipping` followed by coordinates of two points can be used to select a rectangle to draw (zoom). `solution` requests the plotting of the solution by its level lines. The number of level lines to be drawn is selected by `levels` followed by a number.

2.6.2 `InfGraphic`

`InfGraphic` prints a list of the current setting of the parameters given in Table 4.

`boundary`	plot boundary
`solution`	plot level lines
`triangulation`	plot triangulation
`index`	plot index of points
`points`	plot points
`clear`	plot nothing
`clipping`	select a rectangle to plot
`levels`	set number of level lines

Table 4: Parameters of `Graphic`

Example

```
kaskade:graphic clipping (-0.1,0.1,0.1,-0.1) solution
```

```
kaskade:infgraphic
Graph: graphic settings:
Graph:      triangulation rectangle (-0.500,-0.500,0.500,0.500)
Graph:      clipping (-0.100,0.100,0.100,-0.100)
Graph:      to draw level lines
Graph:      7 levels
kaskade:
```

2.6.3 Show [PostScript] [noframe] [continue]

Show draws a picture corresponding to the current setting of Graphic and the current driver. (Up to now only a PostScript driver is available.) A driver might be selected by a parameter (see Table 5).

PostScript	select PostScript driver
continue	hold output (no showpage)
noframe	plot no frame

Table 5: Parameters of Show

Example

```
Kaskade: show noframe
Graph: computed level lines (-3.005e-02 - 0.000e+00)3.756e-03
PS: 454 lines written to PSKaskade0
Kaskade:
```

2.6.4 PostScript [size (⟨*size*⟩)] [origin (⟨*point*⟩)] [font ⟨*fontName*⟩] [file ⟨*fileName*⟩]

PostScript sets the environment for the PostScript driver. The size of the picture and the place on the page might be selected by the parameters size and origin. The values are in units of centimetres. For other parameters see Table 6.

2.6.5 InfPS

InfPS prints a list of the current setting of the parameters of the PostScript driver.

`size`	set size of picture
`origin`	set place of picture
`font`	use font name
`file`	write to file

Table 6: Parameters of `PostScript`

Example

```
kaskade:PostScript file pic.ps
kaskade:InfPS
PS: PostScript settings
PS:     picture size (20.93cm,29.65cm) at (0.00cm,0.00cm)
PS:     pagesize (20.93cm,29.65cm)
PS:     pixelsize  1.312333e-03
PS:     font 'Palatino-Roman'
PS:     file 'pic.ps'
kaskade:
```

2.6.6 `Window` [automatic] [top][bottom][left] [right] [size (⟨*size*⟩)] [origin (⟨*point*⟩)] [font ⟨*fontName*⟩] [name ⟨*windowName*⟩]

`Window` opens a window as a graphic output device. The size and place of the picture can be selected by the `size` and `origin` parameters. The values are given in centimetres. Preselected positions and sizes are `top`,`bottom`, `left`, and `right` allowing four windows on the screen. The current graphic selection (that is what to draw) is saved by using `automatic`. In this case the picture is redrawn automatically, whenever necessary.

2.6.7 `InfWindow`

`InfWindow` prints a list of the current setting of the parameters of the window driver.

Example

```
kaskade:window top right
kaskade:InfWindow
Window: current driver (window) settings
Window:     picture size (18.24cm,12.13cm) at (18.24cm,13.43cm)
```

```
Window:     pagesize (36.47cm,27.57cm)
Window:     resolution 1.000000e+00
Window:     font 'courier'
Window:     (file)name 'Kaskade Graphic'
kaskade:
```

3 Definition of a new problem

To use other functions p_1, p_2, q, and γ than the predefined ones, a change in the KASKADE sources and a recompilation is necessary. Here only a simple recipe is given, for more information see the Programmer's Manual.

The source file `problem.c` contains the functions of some standard problems. The name of the functions with their parameter lists are given in Table 7.

`REAL StdpX(x, y)`	p_1
`REAL StdpY(x, y)`	p_2
`REAL Stdq(x, y)`	q
`REAL Stdg(x, y)`	g
`REAL StdBValue(x, y)`	γ

Table 7: Functions to define the problem

The standard functions are

```
static REAL StdpX(x, y)
  REAL x, y;
  { return actProblem->pXConst; }
static REAL StdpY(x, y)
  REAL x, y;
  { return actProblem->pYConst; }
static REAL Stdq(x, y)
  REAL x, y;
  { return actProblem->qConst; }
static REAL Stdg(x, y)
  REAL x, y;
  { return actProblem->gConst; }
static REAL StdBValue(x, y)
  REAL x, y;
  { return actProblem->BValueConst; }
```

4 An Example

In the following example the Laplace equation is solved on a L–shaped domain. At the end a PostScript picture with the triangulation and the level lines is produced.

The file `L-shape.ksk` contains the following commands.

```
ReadTri L-shape.geo            % reading the triangulation
Refine all                     % refining uniformly
SetBreak globalError 1.0e-2    % should stop, if estimated
                               % error 1.0e-2 is reached
SetBreak dimension 4000        % should stop, if more than
                               % 4000 points
InfSolve                       % checking break conditions
Solve                          % solve until a break condition
                               % holds
  % selecting what to draw: the triangulation
Graphic boundary triangulation
  % defining where to draw: on the left bottom
PostScript file L-shape.ps size (6.0,6.0) origin (0.0,0.0)
  % drawing without newpage
Show continue noframe
  % selecting what to draw: the solution
Graphic clear boundary solution
  % defining where to draw: right beside the triangulation
PostScript origin (6.5,0.0)
  % drawing with newpage
Show continue noframe
```

Then KASKADE will produce the following output.

```
(7)%kaskade
Start of Kaskade 1.0, 22. Feb. 1989
Graph: no triangulation - no graphic
Kaskade: do L-shape.ksk
ReadTri: 8 points, 13 edges, 6 triangles read from L-shape.geo
Solve: state of solution process
  level          1 -    10
  depth          1 -    10
  dimension     21 -  4000
  global error 1.000e+00 - 1.000e-02
```

```
  max steps for Iterate: 100, last iteration error:0.000e+00
  no direct solution available
  no failures yet
  step    level depth  dim    estiErr  iter#    iteErr
Direct      1     1     21
Estimate    2     2     59  5.762e-02
Iterate     2     2     59               3   6.511e-03
Estimate    3     3    198  3.913e-02
Iterate     3     3    198               3   8.059e-03
Estimate    4     4    555  2.426e-02
Iterate     4     4    555               3   6.633e-03
Estimate    5     5   1580  1.540e-02
Iterate     5     5   1580               4   2.857e-03
Estimate    6     6   4371  8.939e-03
Iterate     6     6   4371               3   2.642e-03
BreakReason: global (estimated) error reached
Graph: computed level lines (-3.718e-02 - 0.000e+00)4.647e-03
Kaskade:
```

The file `L-shape.ps` contains the Figure 3.

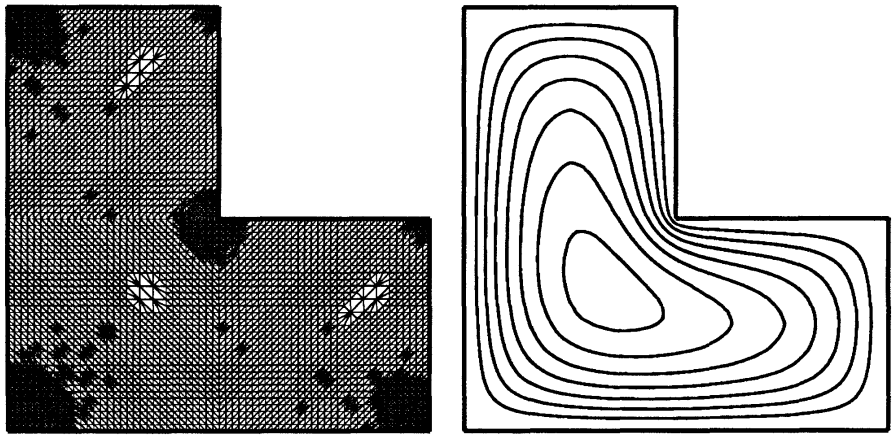


Figure 3: PostScript file `L-shape.ps`

Acknowledgments

The author thanks P. Leinen for the helpful discussions during the development of the program and R. Kornhuber for being the first user with many new requirements.

References

[1] O. Axelsson, V.A. Barker: *Finite Element Solution of Boundary Value Problems: Theory and Computation.* New York: Academic Press (1984)

[2] R.E. Bank: *PLTMG Users' Guide, Edition 5.0.* Technical Report, Department of Mathematics, University of California at San Diego (1988)

[3] P. Deuflhard, P. Leinen, H. Yserentant: *Concept of an Adaptive Hierarchical Finite Element Code.* IMPACT of Computing in Science and Engineering, I, 3-35 (1989)

[4] R. Kornhuber, R. Roitzsch: *On Adaptive Grid Refinement in the Presence of Internal or Boundary Layers.* Preprint, ZIB SC 89–5 (1989)

[5] P. Leinen: *Work done in preparation of a dissertation.* (1989)

[6] R. Roitzsch: *Kaskade Programmer's Manual.* Technical Report, ZIB TR 89–5, Berlin (1989)

Veröffentlichungen des Konrad-Zuse-Zentrum für Informationstechnik Berlin
Preprints August 1989

SC 86-1. P. Deuflhard; U. Nowak. *Efficient Numerical Simulation and Identification of Large Chemical Reaction Systems.*
SC 86-2. H. Melenk; W. Neun. *Portable Standard LISP for CRAY X-MP Computers.*

SC 87-1. J. Anderson; W. Galway; R. Kessler; H. Melenk; W. Neun. *The Implementation and Optimization of Portable Standard LISP for the CRAY.*
SC 87-2. Randolph E. Bank; Todd F. Dupont; Harry Yserentant. *The Hierarchical Basis Multigrid Method. (vergriffen) In: "Numerische Mathematik" Nr. 52, 1988, 427-458.*
SC 87-3. Peter Deuflhard. *Uniqueness Theorems for Stiff ODE Initial Value Problems.*
SC 87-4. Rainer Buhtz. *CGM-Concepts and their Realizations.*
SC 87-5. P. Deuflhard. *A Note on Extrapolation Methods for Second Order ODE Systems.*
SC 87-6. Harry Yserentant. *Preconditioning Indefinite Discretization Matrices.*

SC 88-1. Winfried Neun; Herbert Melenk. *Implementation of the LISP-Arbitrary Precision Arithmetic for a Vector Processor.*
SC 88-2. H. Melenk; H. M. Möller; W. Neun. *On Gröbner Bases Computation on a Supercomputer Using REDUCE.* (vergriffen)
SC 88-3. J. C. Alexander; B. Fiedler. *Global Decoupling of Coupled Symmetric Oscillators.*
SC 88-4. Herbert Melenk; Winfried Neun. *Parallel Polynomial Operations in the Buchberger Algorithm.*
SC 88-5. P. Deuflhard; P. Leinen; H. Yserentant. *Concepts of an Adaptive Hierarchical Finite Element Code.*
SC 88-6. P. Deuflhard; M. Wulkow. *Computational Treatment of Polyreaction Kinetics by Orthogonal Polynomials of a Discrete Variable.* (vergriffen)
SC 88-7. H. Melenk; H. M. Möller; W. Neun. *Symbolic Solution of Large Stationary Chemical Kinetics Problems.*
SC 88-8. Ronald H. W. Hoppe; Ralf Kornhuber. *Multi-Grid Solution of Two Coupled Stefan Equations Arising in Induction Heating of Large Steel Slabs.*
SC 88-9. Ralf Kornhuber; Rainer Roitzsch. *Adaptive Finite-Element-Methoden für konvektions-dominierte Randwertprobleme bei partiellen Differentialgleichungen.*
SC 88-10. S -N. Chow; B. Deng; B. Fiedler. *Homoclinic Bifurcation at Resonant Eigenvalues.*

SC 89-1. Hongyuan Zha. *A Numerical Algorithm for Computing the Restricted Singular Value Decomposition of Matrix Triplets.*
SC 89-2. Hongyuan Zha. *Restricted Singular Value Decomposition of Matrix Triplets.*
SC 89-3. Wu Huamo. *On the Possible Accuracy of TVD Schemes.*
SC 89-4. H. Michael Möller. *Multivariate Rational Interpolation: Reconstruction of Rational Functions.*
SC 89-5. Ralf Kornhuber; Rainer Roitzsch. *On Adaptive Grid Refinement in the Presence of Internal or Boundary Layers.*
SC 89-6. Wu Huamo; Yang Shuli. *MmB-A New Class of Accurate High Resolution Schemes for Conservation Laws in Two Dimensions.*
SC 89-7. U. Budde; M. Wulkow. *Computation of Molecular Weight Distributions for Free Radical Polymerization Systems.*

Veröffentlichungen des Konrad-Zuse-Zentrum für Informationstechnik Berlin
Technical Reports August 1989

TR 86-1. H. J. Schuster. *Tätigkeitsbericht (vergriffen)*

TR 87-1. Hubert Busch; Uwe Pöhle; Wolfgang Stech. *CRAY-Handbuch. - Einführung in die Benutzung der CRAY.*
TR 87-2. Herbert Melenk; Winfried Neun. *Portable Standard LISP Implementation for CRAY X-MP Computers. Release of PSL 3.4 for COS.*
TR 87-3. Herbert Melenk; Winfried Neun. *Portable Common LISP Subset Implementation for CRAY X-MP Computers.*
TR 87-4. Herbert Melenk; Winfried Neun. *REDUCE Installation Guide for CRAY 1 / X-MP Systems Running COS Version 3.3*
TR 87-5. Herbert Melenk; Winfried Neun. *REDUCE Users Guide for the CRAY 1 / X-MP Series Running COS. Version 3.3*
TR 87-6. Rainer Buhtz; Jens Langendorf; Olaf Paetsch; Danuta Anna Buhtz. *ZUGRIFF - Eine vereinheitlichte Datenspezifikation für graphische Darstellungen und ihre graphische Aufbereitung.*
TR 87-7. J. Langendorf; O. Paetsch. *GRAZIL (Graphical ZIB Language).*

TR 88-1. Rainer Buhtz; Danuta Anna Buhtz. *TDLG 3.1 - Ein interaktives Programm zur Darstellung dreidimensionaler Modelle auf Rastergraphikgeräten.*
TR 88-2. Herbert Melenk; Winfried Neun. *REDUCE User's Guide for the CRAY 1 / CRAY X-MP Series Running UNICOS. Version 3.3.*
TR 88-3. Herbert Melenk; Winfried Neun. *REDUCE Installation Guide for CRAY 1 / CRAY X-MP Systems Running UNICOS. Version 3.3.*
TR 88-4. Danuta Anna Buhtz; Jens Langendorf; Olaf Paetsch. *GRAZIL-3D. Ein graphisches Anwendungsprogramm zur Darstellung von Kurven- und Funktionsverläufen im räumlichen Koordinatensystem.*
TR 88-5. Gerhard Maierhöfer; Georg Skorobohatyj. *Parallel-TRAPEX. Ein paralleler, adaptiver Algorithmus zur numerischen Integration ; seine Implementierung für SUPRENUM-artige Architekturen mit SUSI.*

TR 89-1. *CRAY-HANDBUCH. Einführung in die Benutzung der CRAY X-MP unter UNICOS.*
TR 89-2. Peter Deuflhard. *Numerik von Anfangswertmethoden für gewöhnliche Differential-*gleichungen.
TR 89-3. Artur Rudolf Walter. *Ein Finite-Element-Verfahren zur numerischen Lösung von Erhaltungsgleichungen.*
TR 89-4. Rainer Roitzsch. *Kascade User's Manual. Version 1.0*
TR 89-5. Rainer Roitzsch. *Kascade Programmer's Manual. Version 1.0*
TR 89-6. Herbert Melenk; Winfried Neun. *Implementation of Portable Standard LISP for the SPARC Processor.*