Konrad-Zuse-Zentrum für Informationstechnik Berlin
Heilbronner Str. 10, D-10711 Berlin - Wilmersdorf

Rudolf Beck
Bodo Erdmann
Rainer Roitzsch

KASKADE User's Guide

Version 3.x

Technical Report TR 95-11 (December 1995)

KASKADE User's Guide
Version 3.x

Rudolf Beck Bodo Erdmann Rainer Roitzsch

Abstract

KASKADE 3.x was developed for the solution of partial differential equations in one, two, or three space dimensions. Its object-oriented implementation concept is based on the programming language C++ . Adaptive finite element techniques are employed to provide solution procedures of optimal computational complexity. This implies a posteriori error estimation, local mesh refinement and multilevel preconditioning.

The program was designed both as a platform for further developments of adaptive multilevel codes and as a tool to tackle practical problems. Up to now we have implemented scalar problem types like stationary or transient heat conduction. The latter one is solved with the Rothe method, enabling adaptivity both in space and time. Some nonlinear phenomena like obstacle problems or two-phase Stefan problems are incorporated as well. Extensions to vector-valued functions and complex arithmetic are provided.

This report helps to work with KASKADE . Especially we

- study a set of examples,
- explain how to define a user's problem and
- introduce a graphical user interface.

We are extending this guide continuously. The latest version is available by network.

Contents

List of Tables

List of Figures

1 Introduction

The finite element code KASKADE, version 3.x, solves some types of linear elliptic or parabolic equations, and some nonlinear problems. A more detailed description of the related problem classes is given in appendix A.

Our main goal is to present an easy way to customize this software package for solving a wide range of partial differential equations. Of course, there are a lot of applications of similar problem types which cannot be handled directly with the included algorithms. But we hope that our additional advices for extending the code by numerical methods or problem classes will help to use the code as a base for the development of new prototypes.

After reading the first two chapters, the reader should be able to install the code and to run a set of provided examples. In the following chapters he can learn how to implement his own problem.

This user's guide does neither provide a treatment of mathematical concepts nor does it describe the object-oriented implementation of these. Such issues are covered in the technical report [BER95].

We are extending this tutorial continuously. The latest version is available by network as described in the next chapter. If there are any users extending KASKADE , we would be pleased to get notice of their work. If possible, we will publish their solution in the appendix.

The authors are very thankful for any suggestion to improve the code or this guide. Please send your contribution by e-mail to

erdmann@zib-berlin.de or roitzsch@zib-berlin.de.

Please note, that there is another way to exchange experiences about KASKADE : the mailing list `kaskade-l@zib-berlin.de`.

You can subsribe by sending a mail to the listserver `Majordomo@zib-berlin.de` with the message `subscribe kaskade-l` in the body.

By this list you are always informed on essential news about the code and its applications, and you can use this channel for discussing questions of general interest.

2 Installation

2.1 Obtaining the Source Code

The KASKADE source code is part of the `CodeLib`, a collection of numerical codes developed at the Konrad–Zuse–Zentrum. It is available in the electronical library *elib* by anonymous ftp:

```
> ftp elib ( Internet:  130.73.108.11 )
> username:  anonymous
> password:  e-mail-address
> cd pub/kaskade/3.x
```

In this directory you find a compressed tar–file `3.x.tar.Z` and a `README` with some hints for installation. `x` stands for the number of the newest version.

By the commands

```
> binary
> get README
> get 3.x.tar.Z
```

you fetch copies into your local directory.

The tar-file `3.x.tar.Z` contains the source code and examples. Use the following commands to extract these files:

```
> uncompress 3.x.tar.Z
> tar -xf 3.x.tar
```

2.2 Compiling and Linking the Code

The make-file is `kaskade.make`. There are four 'targets' in `kaskade.make` to obtain separate versions for different space dimensions (and one that comprises all of them):

`> make -f kaskade.make k1`	→ 1D-code `k1`
`> make -f kaskade.make k2`	→ 2D-code `k2`
`> make -f kaskade.make k3`	→ 3D-code `k3`
`> make -f kaskade.make k6`	→ code included for all space dimensions

Or shorter, if you have made a copy `makefile` of `kaskade.make`:

`> make k1`	→ 1D-code `k1`
`> make k2`	→ 2D-code `k2`
`> make k3`	→ 3D-code `k3`
`> make k6`	→ code included for all space dimensions

Additionally, the desired space dimension has to be set in the file `dimension.h`. `k6` is the default target.

The default compiler is `GNU's g++, version 2.7.2`. If you prefer `GNU's g++, version 2.6.3`, you have to change the type of *complex* in the file `general.h`. Additionally, we use the `C Set ++ for AIX/6000, version 2.1` on IBM RS/6000 workstations and `DEC C++, V5.0-3` for DEC OSF/1.

2.3 More about the Makefile

The makefile helping to install KASKADE is called `kaskade.make`. It includes the configuration of system resources necessary to compile and link the program. Here you can select your compilers (C++, Fortran), loaders and the pathnames for the libraries you need on your machine (e.g. X11, Fortran,...). We mentioned examples for some platforms. Maybe they are not up to date.

In the makefile you can select your compile and loader options, e.g. `FORFLAGS` or `OPTFLAG`.

Finally, all the file dependencies are listed. Thus we are sure that changes in one of the source files initiate a new compilation. A user is recommended to update this list after he added new dependencies (e.g. include files). That can be done by

```
make -f kaskade.make dependencies
```

generating a new makefile `makefile` with all discovered dependencies.

3 Starting the Program

3.1 Parameters and Parameter Files

Please see section 2 how to obtain the source files and how to compile and link the code.

The name of the program depends on the chosen program version:

1D: `k1`,
2D: `k2`,
3D: `k3`,
all three space dimensions: `k6`.

The program is started by specifying a parameter file, which defines a problem and sets relevant parameters, e.g.

```
> k6 cmd=unit-1d.cmd
```

Here `unit-1d.cmd` is the command file for the selected problem (1D static heat conduction).

Parameters may also be set on the command-line, e.g.

```
> k6 cmd=unit-1d.cmd graphics=1 globalPrecision=1.0e-2
```

Commands have the following syntax:

(1)	`command`	(example: `pause`)
(2)	`command=number`	(example: `spaceDim=3`)
(3)	`command=keyword`	(example: `problem=staticHeatConduction`)

The case (1) is only possible for commands of boolean type and is equivalent to `command=1`, where `1` means true. The value is set to false by `command=0`.

Some general remarks on commands and command files:

- All commands are read at startup. Thus the program execution cannot be altered during run-time (with a few exceptions, e.g. for output routines).
- The symbol '`#`' starts a comment up to the rest of the line in any input file.
- The main command file `kaskade.init` is always read as the first. It sets default values for most commands.
- Parameter values are overwritten by subsequent commands.

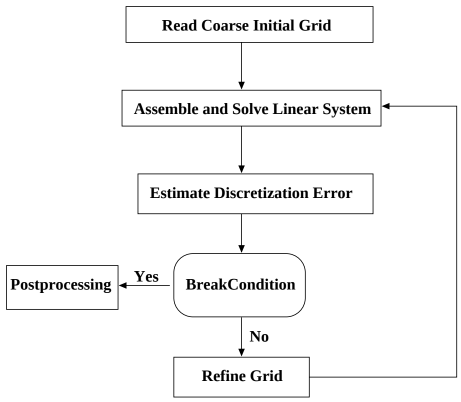

Figure 1: Main iteration loop of the adaptive solution procedure

- If the characters '`<CR>`' appear on the output screen at run-time, the function `Pause()` has been activated by the command `pause` and the program expects input from the user (see the description of `pause` in table 1).

Some important parameters are shown in Table 1.

Any problem (like the one specified by `unit-1d.cmd`) expects a geometry definition file (e.g. `unit-1d.geo`) and a material file (e.g. `unit-1d.mat`).

3.2 Algorithmic Kernel

The finite element methods in KASKADE use adaptive multilevel techniques to achieve of optimal complexity, see [DLY89].

Here we give a short outline of the algorithm in the elliptic case, which is selected by the command `problem=staticHeat`:

1. The user has to define the following input:

- Coefficients of the equation in a `mat-file` for piecewise constant values or he has provide appropriate functions in the source code. The material class is specified by the parameter `material`.

- Dirichlet boundary conditions. The command `DirichletBCs` selects the (constant or variable) Dirichlet boundary conditions.

Parameter Name	**Default** (set in file kaskade.init)	**Description**
`dirichletBCs=`	`constDirichlet`	identifies the type of Dirichlet boundary condition
`file=`	`unit-2d.geo`	name of geometry file (the extension `.geo`) need not be specified. The material file is expected to have the same name, but the extension `.mat`.
`globalPrecision=`	`1e-3`	desired relative precision (maximum discretization error with respect to global energy) if `absPrecision`=0, otherwise desired absolute precision
`material=`	`defaultMaterial`	name identifying the material type, `defaultMaterial` defines constant coefficients on the elements
`linSolver=`	cg	determines the type of iterative solver
`pause=`	`1 (true)`	stop when function `Pause()` is called in the code and wait for input: `<CarriageReturn>` → continue until next `Pause()` is encountered, `'c'` → continue and disable the function `Pause()`, `'p'` → generate a picture in postscript format of the actual mesh and the approximate solution, `'q'` → quit program
`problem=`	`staticHeat`	specifies the problem type to be allocated and solved
`spaceDim=`	`2`	space dimension

Table 1: Some important parameters

- Space dimension. It is set by parameter `spaceDim`.
- A coarse triangulation for the region Ω in a `geo-file`. This file is specified by the parameter `file`.

2. On the actual triangulation of the region Ω the weak formulation of the partial differential equation is discretized by linear finite elements.

The resulting linear system is usually solved by a direct sparse-matrix Cholesky method. This direct sparse solver is only efficient as long as the dimension of the matrix is not too large. We use the command `level0direct=***` to select an iterative solver instead of Cholesky's method, i.e. the direct solver will be used on the first grid (level == 0) for a matrix dimension up to `***`. The switch between direct and iterative solver on higher refinement levels is defined by the command `directSolverLimit=***`, where `***` stands for the maximal matrix dimension for the direct solver on levels > 0.

The iterative solver may include a preconditioner to reduce the number of iterations for the new approximate solution. The iterative solver must be specified with the command `linSolver`, the preconditioner with the command `precond`.

3. The global discretization error of the approximate solution is estimated. If it is below the required precision, the process stops. Otherwise we use local error indicators to refine the mesh where the errors are beyond a threshold.

The estimator is selected by the command `errorEstimator`. A refinement strategy uses the local error information to mark elements for refinement; it can be chosen by the command `refStrategy`.

4. We repeat the steps 2. and 3. until a convergence or break condition stops this cycle. Normally the process ends if the desired precision (defined by command `globalPrecision=***`) or a maximal number of refinement steps (defined by command `maxRefSteps`) is reached. The command `minRefSteps` prescribes a minimal number of refinement steps.

By default, we look at the estimated error relative to the energy norm of the approximate solution. Only if the parameter `absPrecision` is set to `true`, we compute until the absolute error is below the threshold `globalPrecision`. This break condition is reasonable if we want to approximate a solution with a range near zero.

Figure 1 illustrates the main iteration loop of this adaptive process.

The self–adaptive grid refinement minimizes the number of nodes to achieve the requested precision. Additionally, the refinement history allows an efficient multilevel preconditioning of the linear systems.

For linear parabolic partial differential equations we use the adaptive approach of F. Bornemann, see [Bor90], which is able to handle complicated geometries and inconsistent initial data. Here the time is discretized first, allowing an adaptive stepsize control. The errors of the arising spatial elliptic subproblems are controlled independently. Thus advanced adaptive techniques for ordinary differential equations and elliptic boundary value problems are combined.

R. Kornhuber contributes the algorithms for problems formulated as variational inequalities, see appendix A and [Kor95].

3.3 Examples

The KASKADE package includes some example problems, each of them is defined by setting parameters in a command file (extension .cmd). Just type

```
> k6 cmd=****
```

to compute one of them with the executable `k6`. The `****` stands for the name of a command file, e.g.

```
> k6 cmd=unit-1d.cmd
```

In the next chapter you learn how to formulate these command files.

3.3.1 Static Problems

(1) Static problem using the files `peak-1d.cmd`, `peak-1d.geo`, `peak-1d.mat`.

Static heat conduction of type (1) in appendix A on the one-dimensional unit interval.

$$\begin{aligned} -u_{xx} &= f \qquad \text{in } [0,1] \\ u &= 0 \qquad \text{in the points 0 and 1} \end{aligned}$$

This Poisson problem has the constant coefficient $k = 1$ in the Laplacian and a source function $f \sim exp(-100.0 * (x - 0.5)^2)$ with a central peak and zero values on the boundary points.

The solver stops when the relative global error of the approximated solution is less than $1.0e - 5$ (relative to the energy norm). The conjugate gradient method that solves the linear systems is preconditioned by a multigrid algorithm.

The adaptive refinement is tuned by an error estimator. Figure 2 shows the final grid.

In this and in the two following examples we know the true solution of the problems, and by setting the parameter `compareSolution=1` we can compute

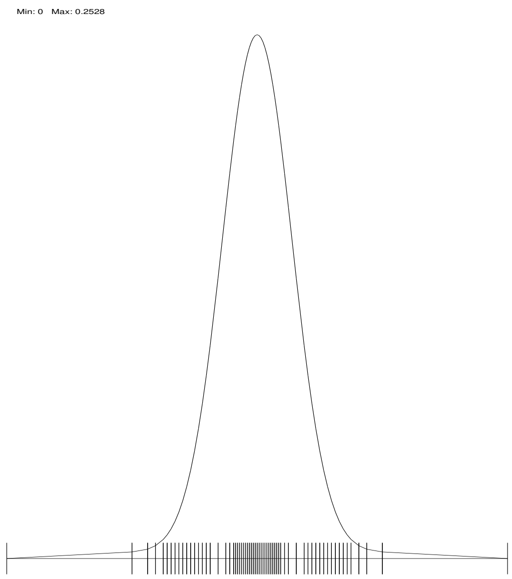

Figure 2: Final mesh and solution in example (1)

the maximum error of the approximate solution in the nodes. In the chapter 8 we explain how you can compare the approximate solution of your problem with a given function (e.g. the true solution).

(2) Static problem using the files `peak-2d.cmd`, `peak-2d.geo`, `peak-2d.mat`.

This problem is the 2D–analogue to problem (1).

$$\begin{aligned} -(u_{xx} + u_{yy}) &= f \qquad \text{in } [0,1] \text{ x } [0,1] \\ u &= 0 \qquad \text{on the boundary} \end{aligned}$$

Here the source function $f \sim exp(-100.0 * [(x-0.5)^2 + (y-0.5)^2])$ has a peak in the center of the unit square.

The inital grid consists of only four triangles. In figure 3 you see the grid after 6 refinement steps and the solution in a quasi-3d-plot.

(3) Static problem using the files `peak-3d.cmd`, `peak-3d.geo`, `peak-3d.mat`.

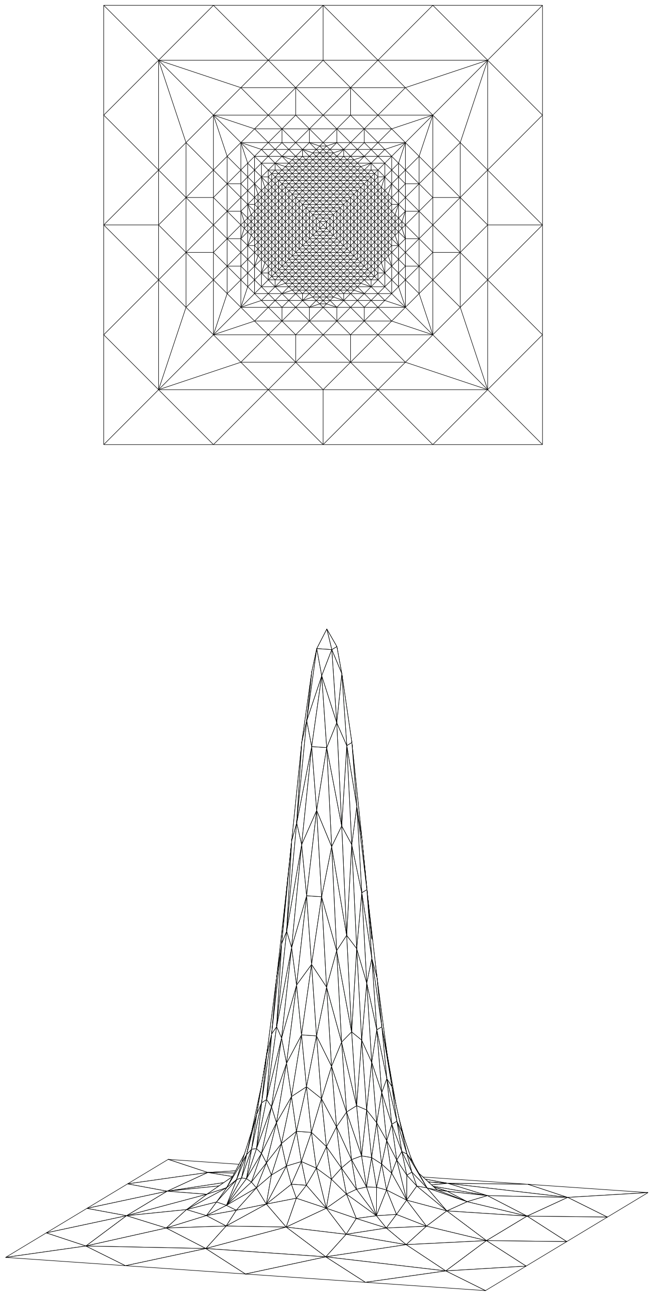

Figure 3: Mesh and quasi-3d-plot of the solution after 6 refinement steps in example (2)

This problem is the 3D–analogue to problem (1).

$$\begin{aligned} -(u_{xx} + u_{yy} + u_{zz}) &= f \qquad \text{in } [0,1] \text{ x } [0,1] \text{ x } [0,1] \\ u &= 0 \qquad \text{on the boundary} \end{aligned}$$

Here the source function $f \sim exp(-100.0 * [(x-0.5)^2 + (y-0.5)^2 + (z-0.5)^2])$ has a peak in the center of the unit cube.

The algorithm starts with a partition of the cube into 6 tetrahedra. In figure 4 we show a cut through the finite element mesh provided after some steps of the adaptive solver . This picture and some of the following were produced by GRAPE [1].

For 3D–geometries there is no online–graphics integrated in KASKADE . We recommend to use one of the popular visualization programs GRAPE, EXPLORER[2], or AVS[3] and to use our interfaces to write data on a file in an appropriate format.

(4) Static problem using the files `unit-1d.cmd`, `unit-1d.geo`, `unit-1d.mat`.

Poisson equation with constant coefficients on the unit interval

$$\begin{aligned} -0.0001u_{xx} &= 5.5 \qquad \text{in } [0,1] \\ u &= 0 \qquad \text{in the points 0 and 1} \end{aligned}$$

The coefficient of u_{xx} is arbitrary and chosen so mysteriously only to illustrate the use of keywords (e.g. `Factors`) in the `.mat` file. Compare the chapter 5 and the file `unit-1d.mat`.

(5) Static problem using the files `unit-2d.cmd`, `unit-2d.geo`, `unit-2d.mat`.

Poisson equation with constant coefficients on the unit square

$$\begin{aligned} -(u_{xx} + u_{yy}) &= 1 \qquad \text{in } [0,1] \text{ x } [0,1] \\ u &= 10.0 \qquad \text{on the boundary} \end{aligned}$$

(6) Static problem using the files `unit-2d-a.cmd`, `unit-2d-a.geo`, `unit-2d-a.mat`.

[1]GRAPE, Graphical Programming Environment, Copyright *Sonderforschungsbereich 256, University of Bonn, Germany*

[2]Copyright *The Numerical Algorithms Group Ltd., Oxford UK*

[3]Copyright *Advanced Visual Systems Inc.*

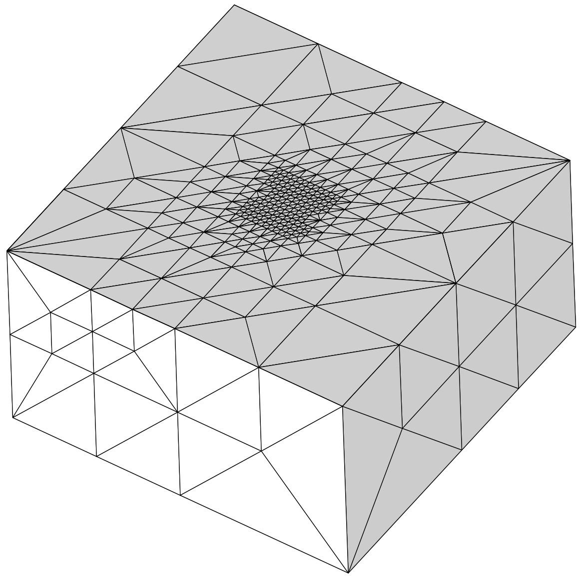

Figure 4: Cut through the adaptive mesh in example (3)

Here we solve the same elliptic equations as in example (5) but with additional Neumann and Cauchy boundary conditions:

$$\begin{aligned} u &= 10.0 && \text{on the boundary } x=0 \text{ or } y=0 \text{ (Dirichlet)} \\ u_y &= 0.0 && \text{on the boundary } y=1 \text{ (Neumann)} \\ u_x + u &= 2.0 && \text{on the boundary } x=1 \text{ (Cauchy)} \end{aligned}$$

Figure 5 illustrates the grid and solution after some refinement steps.

(7) Static problem using the files `slit-2d-45.cmd`, `slit-2d-45.geo`, `slit-2d-45.mat`.

The problem is defined on the polygonial region in figure 6 by the same equation as in example (5) and the boundary conditions

$$\begin{aligned} \partial u/\partial n &= 0.0 && \text{on the boundary } y=0, x>=0 \text{ (Neumann)} \\ u &= 1.0 && \text{elsewhere on the boundary (Dirichlet)} \end{aligned}$$

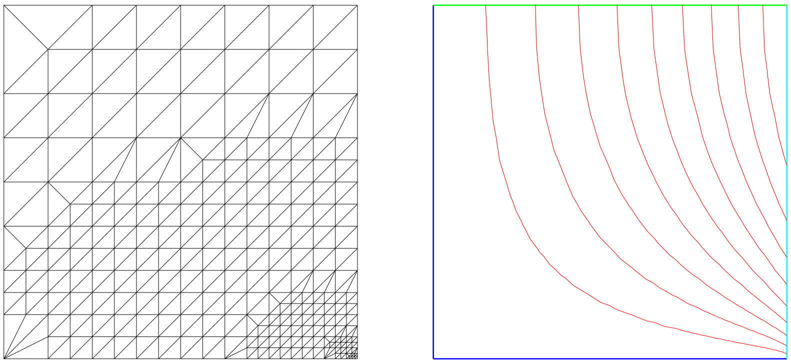

Figure 5: Mesh and solution after some refinement steps in example (6)

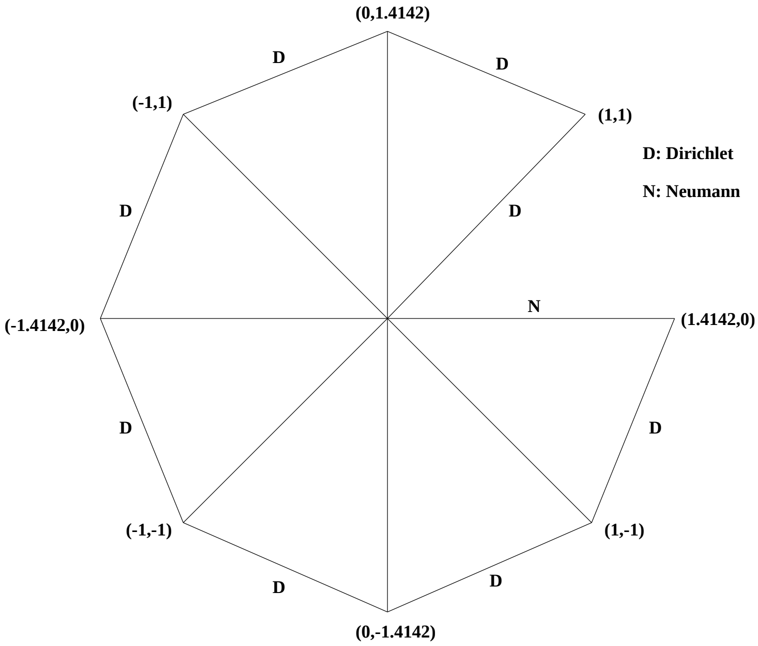

Figure 6: Initial triangulation in example (7)

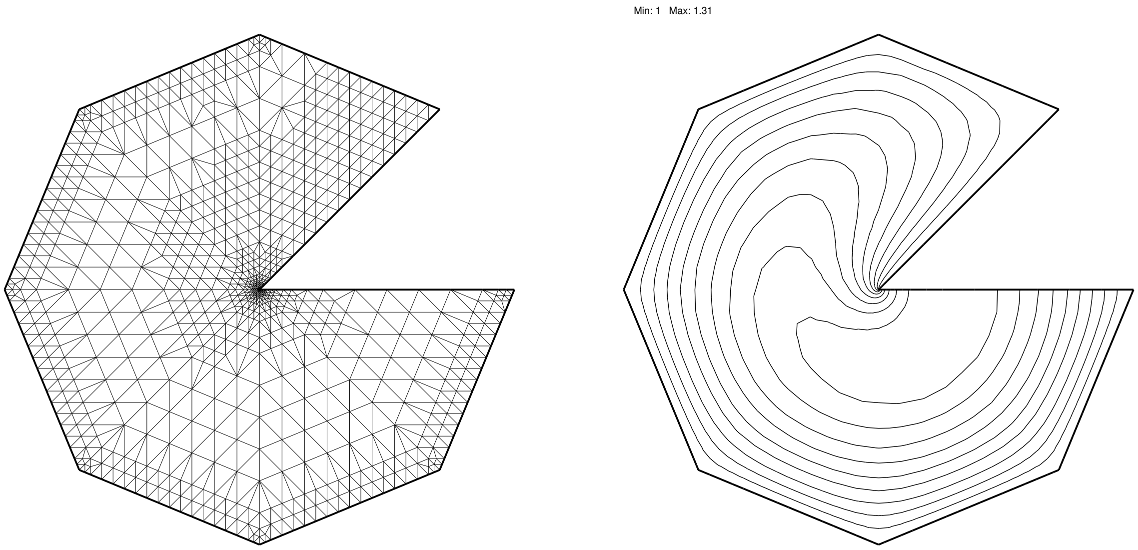

Figure 7: Mesh and solution after 13 refinement steps in example (7)

The adaptive refinement generates a fine mesh in the neighbourhood of the origin where the solution has a singularity.

In figure 7 we show the mesh and the isolines of the solution after 13 refinement steps.

(8) Static problem using the files `slit-2d.cmd`, `slit-2d.geo`, `slit-2d.mat`.

The same problem as in (7) but on a different domain. Here the angle of the slit has been set to 0 degree. That means we have two edges from $(0,0)$ to $(1.4142, 0)$, one with Dirichlet and one with Neumann condition, compare figure 8. Additionally, we approximate a circular boundary during the refinement process.

$$\begin{aligned} \partial u/\partial n &= 0.0 && \text{on the boundary } y = 0, x >= 0 \text{ (Neumann)} \\ u &= 1.0 && \text{elsewhere on the boundary (Dirichlet)} \end{aligned}$$

In figure 9 you see the mesh and the isolines of the solution after adaptive refinement.

(9) Static problem using the files `slit-2d-a.cmd`, `slit-2d-a.geo`, `slit-2d-a.mat`.

Consider Laplace's equation

$$-\Delta u = 0.0$$

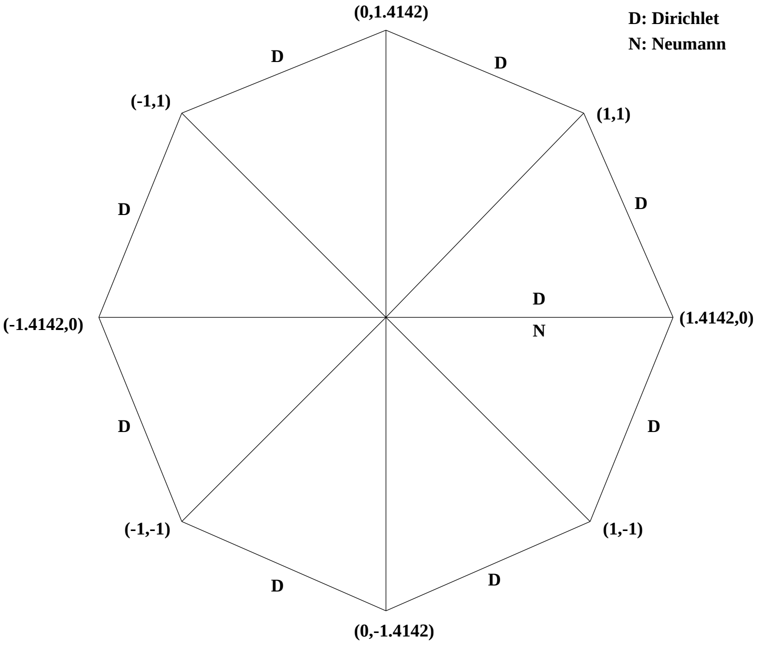

Figure 8: Initial triangulation in example (8)

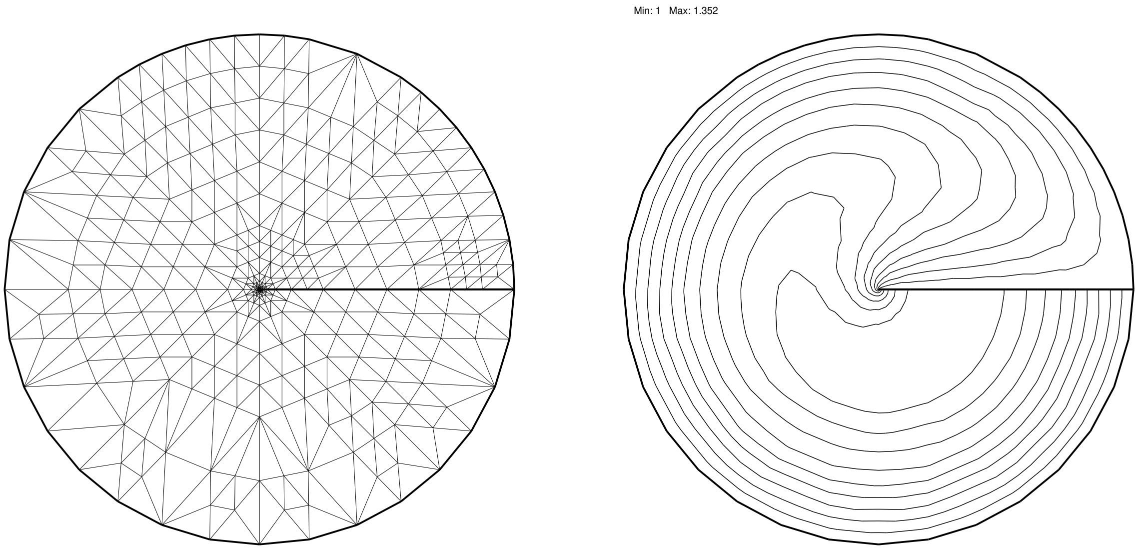

Figure 9: Mesh and solution after adaptive refinement steps in example (8)

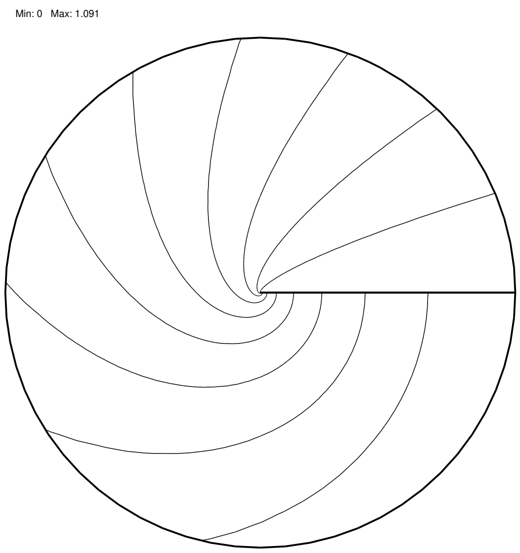

Figure 10: Contour lines of the solution in example (9)

on a circle of radius 1 (centered at the origin) with a slit along the positive x-axis. Homogeneous Dirichlet boundary conditions are imposed on the top, and homogeneous Neumann boundary conditions at the bottom of the slit.

The solution is singular and has the leading term

$$u = r^{1/4} sin(\phi/4).$$

This function is used as Dirichlet boundary condition on the remaining part of the boundary, yielding it as exact solution. The associated contour plot is given in figure 10.

Starting with a simple polygonial region we approximate the circular boundary during the refinement process by placing new boundary points onto the circle.

In figure 11 the meshes after 4, 9, and 12 refinement steps are presented.

(10) Static problem using the files `unit-3d.cmd`, `unit-3d.geo`, `unit-3d.mat`.

Poisson equation with constant coefficients on the unit cube

$$\begin{aligned} -(u_{xx} + u_{yy} + u_{zz}) &= 1 \qquad \text{in } [0,1] \text{ x } [0,1] \text{ x } [0,1] \\ u &= 0 \qquad \text{on the boundary} \end{aligned}$$

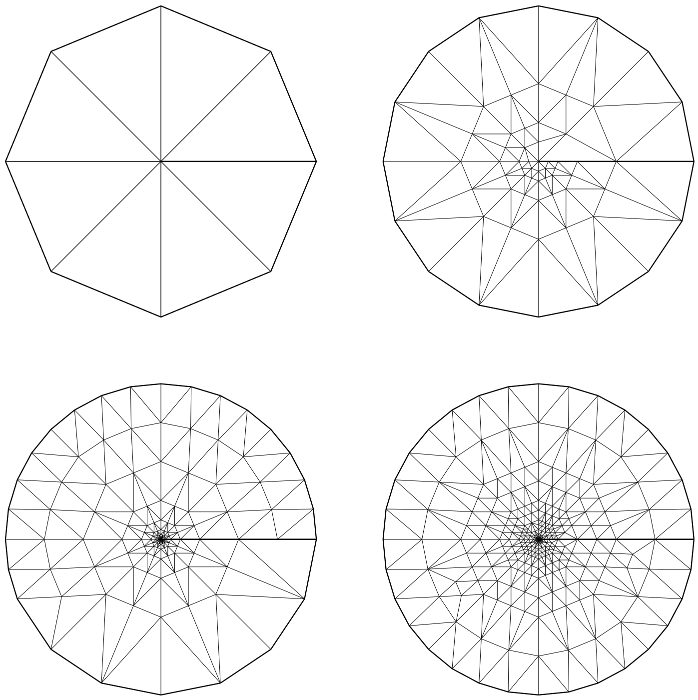

Figure 11: Initial mesh and meshes after 4, 9, and 12 refinement steps in example (9)

(11) Static problem using the files `l-3d.cmd`, `l-3d.geo`, `l-3d.mat`.

The same equation as in Problem (10), but on a region with reentrant corner as described in `l-3d.geo`, see figure 12.

The singularity of the solution in the re-entrant corner is well resolved by the error estimator.

(12) Static problem using the files `jump.cmd`, `jump.geo`, `jump.mat`.

The same equation and region as in Problem (5), but with a jump in the coefficients.

$$\begin{aligned} -100(u_{xx}+u_{yy}) &= 1 && \text{in } [0.25, 0.75] \text{ x } [0.25, 0.75] \\ -(u_{xx}+u_{yy}) &= 1 && \text{elsewhere in the unit square} \\ u &= 10.0 && \text{on the boundary} \end{aligned}$$

In this example it is recommended to work with the option `plot3d=1` for graphical illustration. Figure 13 shows the initial coarse mesh and the solution as

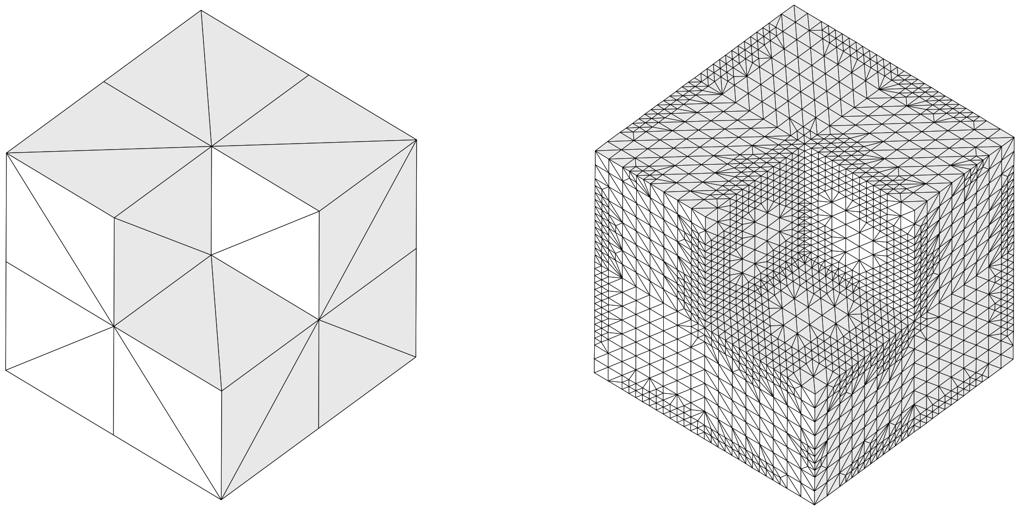

Figure 12: Initial mesh and mesh after some refinement steps in example (11)

quasi–3d plot after some adaptive refinement steps, and in Figure 14 you can see the mesh and solution after seven refinement steps.

(13) Static problem using the files `layer-2d.cmd`, `layer-2d.geo`, `layer-2d.mat`.

Here we want to solve the same equation as in Problem (12). But additionally, we consider a term of type (3) in the weak formulation of the problem, see appendix A. For example, problems of this kind appear in electrodynamics.

$$
\begin{aligned}
-100(u_{xx} + u_{yy}) &= 1 \qquad \text{in } [0.25, 0.75] \text{ x } [0.25, 0.75] \\
-(u_{xx} + u_{yy}) &= 1 \qquad \text{elsewhere in the unit square} \\
u &= 10.0 \qquad \text{on the boundary}
\end{aligned}
$$

On the layer between the two materials (which must be marked by type 1 and 2 in the `.mat`-file) we integrate the term $(k_1 - k_2)(x + y)$, where the material coefficients are $k_1 = 1$ and $k_2 = 100$.

This integration must be initiated by the command `innerBoundary=1`.

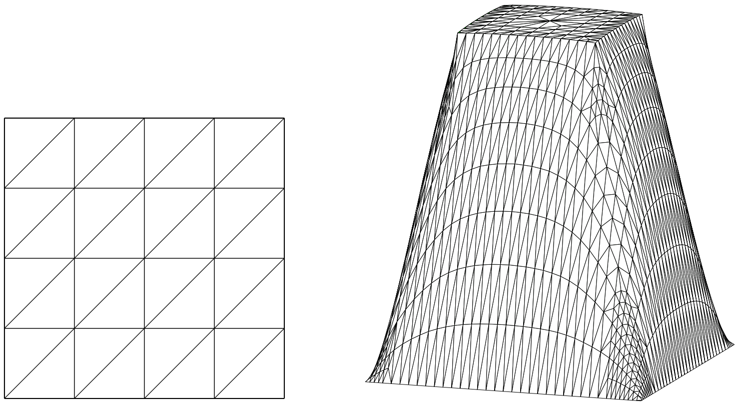

Figure 13: Initial mesh and solution plot after some refinement steps in example (12)

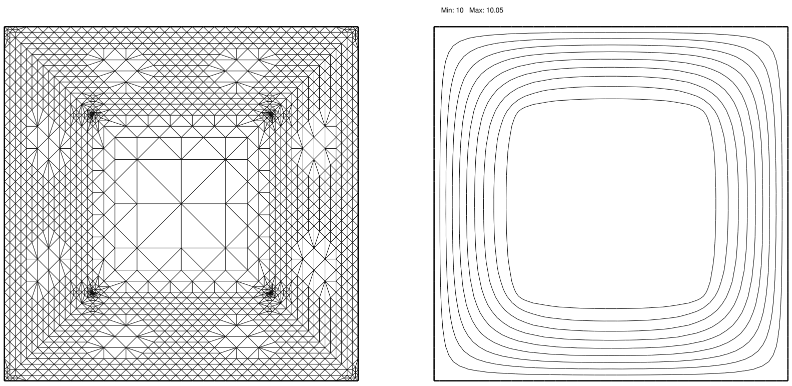

Figure 14: Mesh and solution after seven refinement steps in example (12)

(14) Static problem using the files `layer-3d.cmd`, `layer-3d.geo`, `layer-3d.mat`.

Here we solve the analogous equation to Problem (13) in three space dimensions.

$$\begin{aligned} -100(u_{xx} + u_{yy} + u_{zz}) &= 1 && \text{in } [0.25, 0.75] \text{ x } [0.25, 0.75] \text{ x } [0.25, 0.75] \\ -(u_{xx} + u_{yy} + u_{zz}) &= 1 && \text{elsewhere in the unit cube} \\ u &= 10.0 && \text{on the boundary} \end{aligned}$$

By the command `innerBoundary=1` we additionally integrate the term $(k_1 - k_2)(x + y + z)$ on the layer between the two different materials (which must be marked by type 1 and 2 in the `.mat`-file). $k_1 = 1$ and $k_2 = 100$ denote the material coefficients.

(15) Static problem using the files `flow-2d.cmd`, `flow-2d.geo`, `flow-2d.mat`.

The same equation and region as in Problem (5), but with Neumann boundary conditions on each boundary edge. Only on two points we prescribe Dirichlet values.

$$\begin{aligned} -(u_{xx} + u_{yy}) &= 1 && \text{in } [0, 1] \text{ x } [0, 1] \\ \partial u / \partial n &= 0.0 && \text{on the boundary} \\ u &= 1.0 && \text{in (0,0)} \\ u &= 5.0 && \text{in (1,0)} \end{aligned}$$

Figure 15 shows the initial coarse mesh and the mesh after some adaptive refinement steps.

(16) Static problem using the files `flow-2d-a.cmd`, `flow-2d-a.geo`, `flow-2d-a.mat`.

Here we solve an elliptic equation with a mass term and only Neumann condition on the boundary.

$$\begin{aligned} -(u_{xx} + u_{yy}) + u &= 1 && \text{on a circle of radius } \sqrt{2} \\ \partial u / \partial n &= 1.0 && \text{on the boundary} \end{aligned}$$

Figure 16 shows the mesh and the isolines of the approximate solution after some refinement steps. In chapter 5 we explain how we handle arcs of a circle as boundaries.

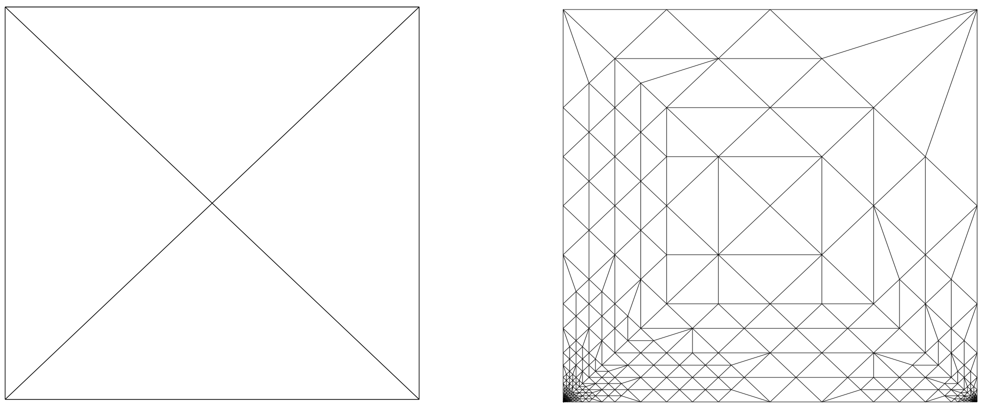

Figure 15: Initial mesh and mesh after some refinement steps in example (15)

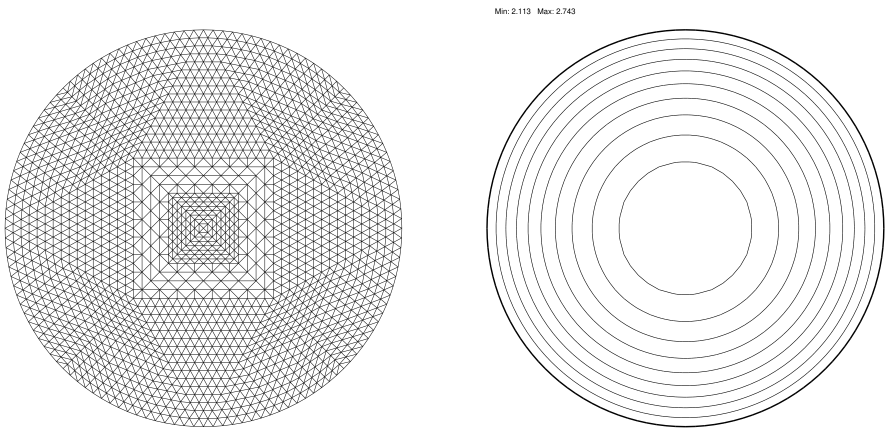

Figure 16: Mesh and contour map of the solution in example (16)

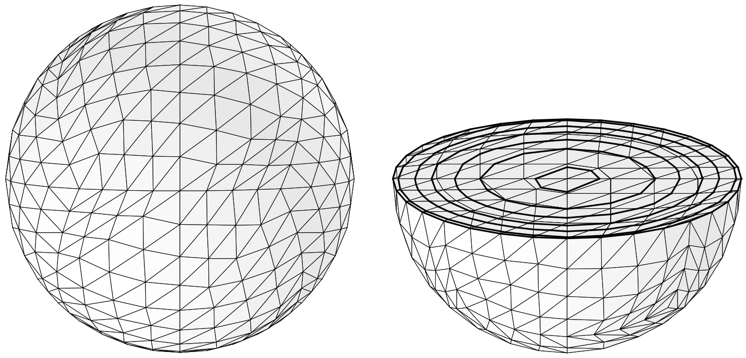

Figure 17: Mesh and contour map of the solution on a cutting plane in example (17)

(17) Static problem using the files `flow-3d.cmd`, `flow-3d.geo`, `flow-3d.mat`.

As in example (16) we solve an elliptic equation with a mass term and only Neumann condition on the boundary.

$$\begin{aligned} -(u_{xx}+u_{yy}+u_{zz})+u &= 1 \qquad \text{in a ball of radius } \sqrt{3} \\ \partial u/\partial n &= 1.0 \qquad \text{on the boundary} \end{aligned}$$

Figure 17 shows the mesh and the isolines of the approximate solution on a cutting plane after some refinement steps. In the chapter 5 about geometrical input we explain how we handle curvilinear boundaries.

(18) Static problem , a calculation from Hyperthermia. In order to demonstrate the applicability of our code to real life problems, we consider a problem arising in hyperthermia.

Hyperthermia is a cancer therapy based on the observation that local heating may slow down the growth of a tumor, especially if it is applied in combination with other methods like chemotherapy or radiotherapy. The heating of the tissue is obtained by radiowaves generated by an antenna array. The antennas are either fixed on the skin or implanted in the tissue itself. Of course, the

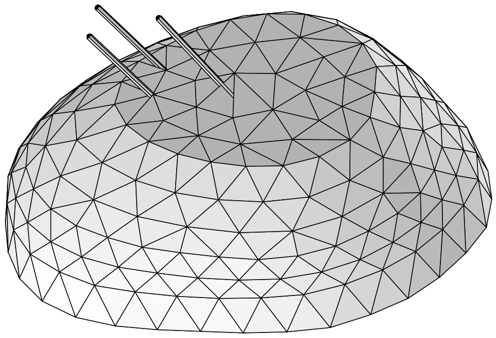

Figure 18: Surface of the initial triangulation of a skull (18)

position of the antennas and frequency of the electric field have to be chosen properly to achieve a local heating of the tumor, compare [HYP89]. This requires the efficient and robust solution of the stationary or transient BHT (bio heat transfer) equation [Pen48]:

$$-\nabla(k\nabla T) + qT = Q_e$$

with coefficients k and q depending of the tissue, Q_e describing the power of the electrical field generated by radiowaves in the tissue. We refer to Seebaß, [See90], for a more detailed explanation and the numerical data of the following example.

We consider the heating of the tumor by the electric field resulting from three implanted antennas. A suitable description of the computational domain Ω is obtained by computer tomography. Figure 18 shows the surface of Ω and the initial triangulation $\mathcal{T}_0$ from Sebaß, [See90]. Note that $\mathcal{T}_0$ is chosen such that different tissues as bone, brain and tumor belong to different tetrahedra.

The adaptive refinement concentrates on the neighborhood of the antennas and the tumor in the interior of the skull where we have high gradients of the temperature. This is illustrated by the figure 19 showing the final triangulation $\mathcal{T}_5$ and the contour lines of the temperature on some clipping plane. The different tissues are indicated by black (bone), white (brain) and grey (tumor inside the brain).

(19) Static problem using the files `cylindrical-3d.cmd`, `cylindrical-3d.geo`, `cylindrical-3d.mat`.

Poisson equation with constant coefficients in a cylindrical region $Z = Z(r, \phi, z)$ with radius $r \in [0, \sqrt{2.0})$ and $z \in [-1, 1]$.

$$-(u_{xx} + u_{yy} + u_{zz}) = -(4.0 + x^2 + y^2) * e^{-z} \text{ in the cylinder Z}$$

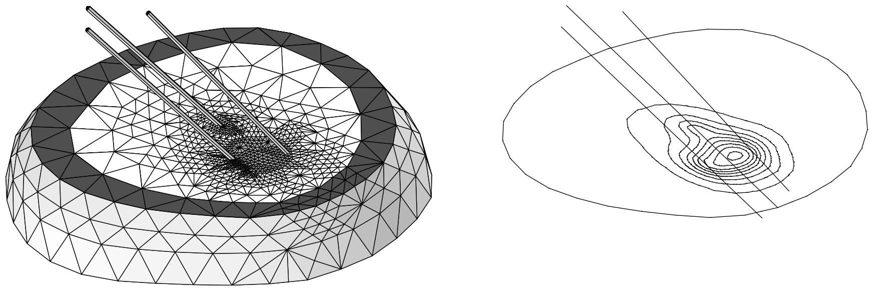

Figure 19: Clipping of the mesh on refinement level 7 and temperature on this plane in example (18)

$$u = (x^2 + y^2) * e^{-z} \qquad \text{on the boundary}$$

We start on a coarse triangulation of the cube $(-1, 1)$ x $[-1, 1]$ x $[-1, 1]$. During refinement we approximate the shape of Z by placing new point coordinates on the surface of the cylinder. Figure 20 shows the first three steps of uniform refinement and figure 20 the mesh of the cylinder and isolines of the solution on a vertical cut through it.

(20) Static problem using the files `cylindrical-2d.cmd`, `cylindrical-2d.geo`, `cylindrical-2d.mat`.

Poisson equation in a cylindrical domain Z formulated in cylindrical coordinates r, ϕ, and z. The cylinder $Z = Z(r, \phi, z)$ has radius $r = 1$ and height z between -1 and 1.

$$\begin{aligned} -\Delta T &= e^{-z} \qquad \text{with} \\ \Delta T &= \frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial T}{\partial r}) + \frac{1}{r^2}\frac{\partial^2 T}{\partial \phi^2} + \frac{\partial^2 T}{\partial z^2} \qquad \text{in Z} \end{aligned}$$

The boundary conditions are shown in figure 21. On the top and on the curvilinear faces we have a temperature of $50°C$. On the bottom there is an area $(r <= 0.5)$ with temperature of $200°C$, and $50 + 300(1 - r)°C$ elsewhere.

This problem has axial symmetry and can be treated as a 2d–problem as described in figure 22:

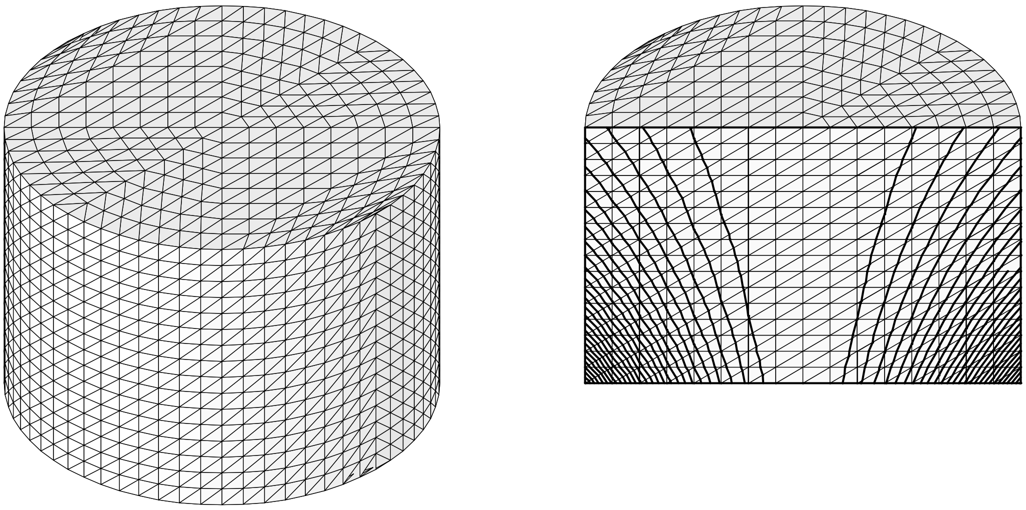

Figure 20: Mesh and solution on a cut after some refinement steps in example (19)

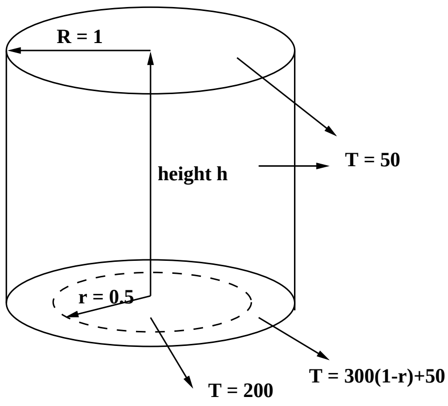

Figure 21: Axialsymmetric problem in example (20)

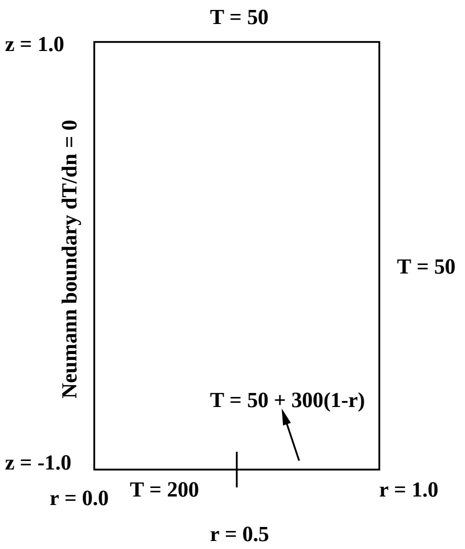

Figure 22: Axialsymmetric problem in example (20)

$$
\begin{aligned}
\frac{\partial T}{\partial \phi} &= 0 \\
\Longrightarrow \qquad \Delta T &= \frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial T}{\partial r}) + \frac{\partial^2 T}{\partial z^2} \qquad \text{in Z} \\
\Longrightarrow \qquad r\Delta T &= \frac{\partial}{\partial r}(r\frac{\partial T}{\partial r}) + \frac{\partial}{\partial z}(r\frac{\partial T}{\partial z}) = re^{-z}
\end{aligned}
$$

By substitutions $r \longrightarrow x$ and $z \longrightarrow y$ this results in the usual 2d–problem with a variable elliptic coefficient $k = x$ and the right-hand side multiplied by x :

$$
-\frac{\partial}{\partial x}(x\frac{\partial T}{\partial x}) - \frac{\partial}{\partial y}(x\frac{\partial T}{\partial y}) = xe^{-y}
$$

At the inner boundary, we have the Neumann boundary condition $\partial T/\partial n = 0$.
Analogously we can treat the Laplace operator in spherical coordinates

$$
\Delta = \frac{1}{r^2}\frac{\partial}{\partial r}(r^2\frac{\partial}{\partial r}) + \frac{1}{r^2 cos^2\theta}\frac{\partial^2}{\partial \phi^2} + \frac{1}{r^2 cos\theta}\frac{\partial}{\partial \theta}(cos\theta\frac{\partial}{\partial \theta})
$$

Multiplying with r^2 and $cos\theta$ yields

$$r^2 cos\theta\Delta = \frac{\partial}{\partial r}(r^2 cos\theta\frac{\partial}{\partial r}) + \frac{\partial}{\partial\phi}(\frac{1}{cos\theta}\frac{\partial}{\partial\phi}) + \frac{\partial}{\partial\theta}(cos\theta\frac{\partial}{\partial\theta})$$

Again, by a simple multiplication of the transformed partial differential equation with a special factor depending on the used coordinate system, we arrive at the standard problem formulation (1), see appendix A. Note that this conversion has also to be done for the Cauchy boundary conditions.

3.3.2 Transient Problems

The following examples demonstrate the parabolic solver, which is adaptive in space and time. In each time step, the computation of the spatial solution starts from the initial grid.

(1) Transient Problem using the files `u-1d-step.cmd`, `u-1d-step.geo`, `u-1d-step.mat`.

The solution of this parabolic problem shows a transient diffusion process starting from a sharp initial profile

$$1000u_t - \Delta u + 100u = 0.0$$

$$u(0) = \begin{cases} 1.0, & \text{for } x \leq 0.5 \\ 0.0, & \text{for } x > 0.5 \end{cases}$$

The coefficients are described in the file `u-1d-step.mat`, the domain (unit interval) in u-1d-step.geo.

The boundary conditions are u=0 in x=0 and u=1 in x=1.

The initial values are provided by a function in the file dirichlettr.cc.

(2) Transient Problem using the files `u-2d-step.cmd`, `u-2d-step.geo`, `u-2d-step.mat`.

Analogous to example (1), but on the unit square. The boundary condition are u=0 on the left side, u=1 opposite to it, and homogeneous Neumann condition on the other parts of the boundary.

(3) Transient Problem using the files `u-3d-step.cmd`, `u-3d-step.geo`, `u-3d-step.mat`.

Analogous to example (2), but on the unit cube.

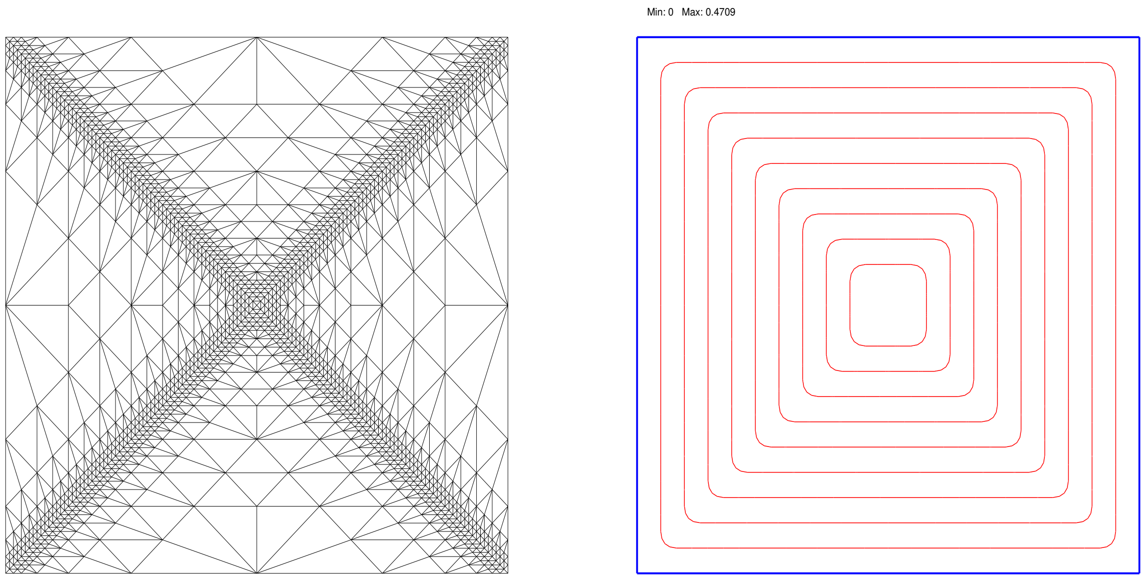

Figure 23: Final mesh and isolines of solution after 8 refinement steps in example (1)

3.3.3 Nonlinear Problems

(1) Nonlinear Problem using the files `obstacle.cmd`, `obstacle.geo`, `obstacle.mat`.

Here we treat a stationary 2D–problem of type (5) in appendix A with constraints $u(x) \leq dist(x, \partial\Omega)$, see figure 23. This obstacle problem is modelling a form of elasto–plastic torsion of a cylindrical bar with cross–section Ω.

The region along the diagonals of Ω is inactive, i.e. not on the obstacle.

Figure 23 shows the final mesh and the contour lines of the solution after 8 adaptive refinement steps.

(2) Nonlinear Problem using the files `stefan.cmd`, `stefan.geo`, `stefan.mat`.

The two–dimensional Stefan problem (6) in appendix A describes the transition between two phases of a material, e.g. a melting iceberg.

Two–phase Stefan problems of this form are arising in the simulation of the heat conduction in a substance undergoing a change of pase at the nominal phase change temperature θ_1. In this case, the function U is a generalized temperature resulting from a standard Kirchhoff transformation $U = \kappa_0$ for $\theta < \theta_1$ and $U = \kappa_1\theta$ for $\theta > \theta_1$ of the physical temperature θ.

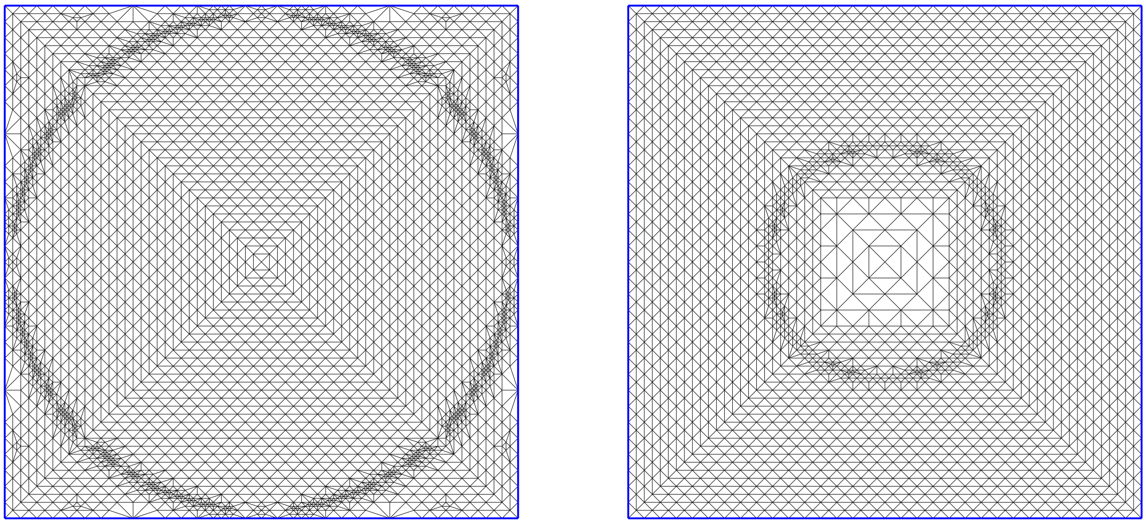

Figure 24: Final triangulations for the first and the last time step in example (2)

There are further applications in metallurgy, soil mechanics, decision and control theory, etc. For more details, we refer to Crank [Cr88], Elliot and Ockendon [EO82], Hoffmann and Sprekels [HS90], Kornhuber [Kor95] and the literature cited therein.

For discretization in time we use a backward Euler scheme with a uniform step-size. The spatial problems at the different time levels are similar to those solved in the obstacle problem (1).

In figure 24 you see the final triangulations for the first and the last time step.

.

(3) Nonlinear Problem using the files `pmedia.cmd`, `pmedia.geo`, `pmedia.mat`.

In this example we consider the degenerate parabolic initial-boundary value problem

$$
\begin{array}{rcl}
\partial\theta/\partial t - \Delta\theta^m = 0, \quad \theta \geq 0, \quad m > 1 & & \text{in} \quad \Omega \times (0,T) \\
\theta(0) = \theta^0 & & \text{in } \Omega \\
\theta = g \quad on \quad \Gamma_D \times (0,T), \quad \partial\theta/\partial n = 0 & & \text{on} \quad \Gamma_N \times (0,T)
\end{array}
$$

of type 6 in appendix A, describing the adiabatic flow of a homogeneous gas through a porous medium [Mus37]. This example in two space dimensions shows the time–dependent diffusion in porous media.

In our example we select $m = 2$ and the initial condition θ^0,

$$\theta^0(x_1, x_2) = [0.4 - r^2(1 + 0.5sin(14\phi))]_+^{1/2},$$

using $r = (x_1^2 + x_2^2)^{1/2}$ and $\phi = atan(x_2/x_1)$. We solve on $\Omega = (0,1) \times (0,1)$ and $T \in (0, 0.05)$ prescribing homogeneous Neumann conditions at $x_1 = 0, x_2 = 0$ and homogeneous Dirichlet conditions at $x_1 = 1, x_2 = 1$.

Again we use only a fixed stepsize for time discretization.

4 Problem Classes

The program handles several problem types. Anyone of these is selected by the command

```
problem=***  ,
```

where `***` represents one of the keywords in the left column of table 2.

The problem type is used internally to describe the structure of the differential equation and to select the suitable solver routines.

Keyword	**problems**
`staticHeat`	linear elliptic problem, e.g. static heat conduction
`transientHeat`	linear parabolic problem, e.g. transient heat conduction
`obstacle`	obstacle problem
`stefan`	Stefan problem
`porousMedia`	porous media equation

Table 2: Keywords for problem types

See appendix A for a desription of basic problem classes.

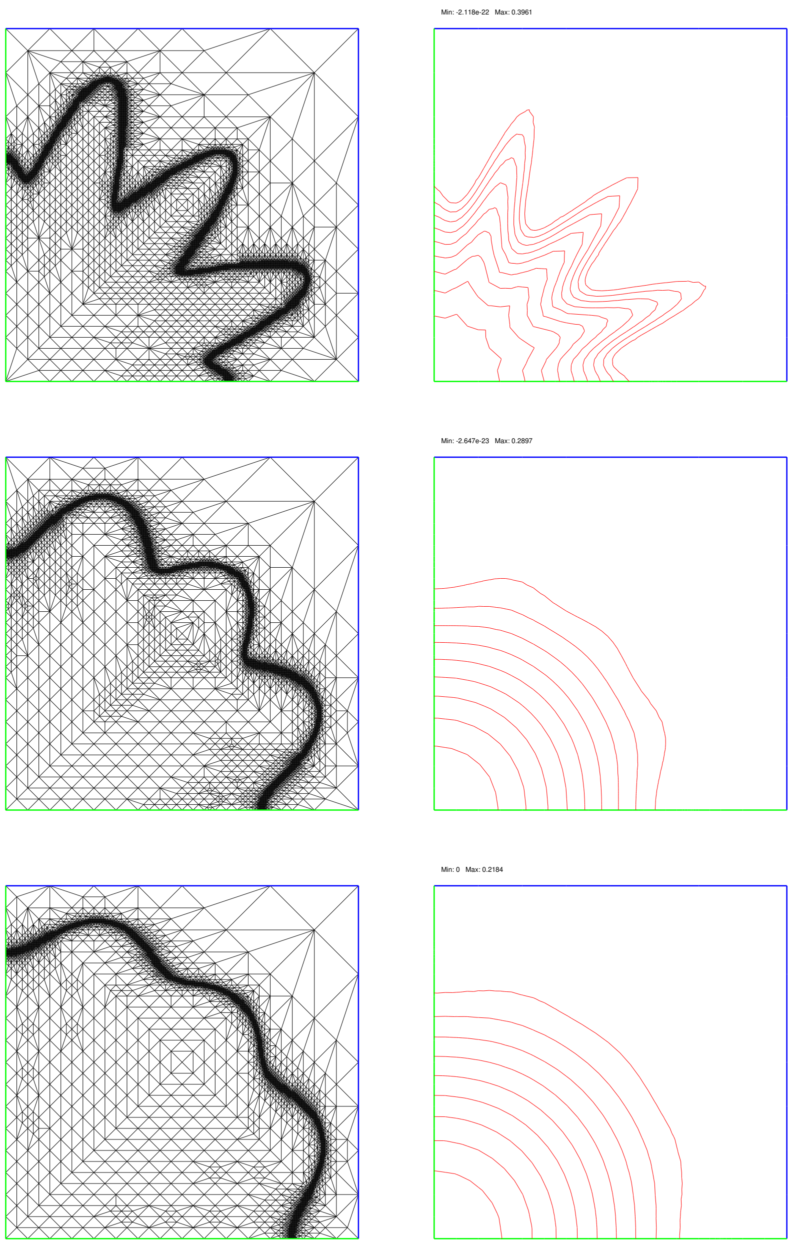

Figure 25: Final mesh and isolines of solution after 1, 20 and 40 time steps in example (3)

5 Geometry and Material Properties

For a complete definition of a problem, we have to specify the type of the partial differential equation (see preceding chapter), the space dimension of the domain, a triangulation of the domain, and the coefficients (e.g. k and f in appendix A, (1)), which usually describe material properties.

5.1 Definition of the Geometry

The space dimension is selected by the command

`spaceDim=***` .

Here `***` stands for 1, 2, or 3. The default value is 2.

The triangulation of the domain Ω is given in a file defining a set of intervals (1D), triangles (2D) or tetrahedra (3D). We use the extension `.geo` for this file. The filename is specified by the command

`file=***` .

We offer two formats to code the geometrical data. By the command `readOldFormat=1` you select the format we used in former versions (up to 2.x) of KASKADE, see Roitzsch et al. [Roi93]. We recommend as default a simpler format (`readZIBFormat=1`) which we describe here:

- 1D–region:

  \# comments
  Points: maxIndex *maxIndex*
  <index of point >   *<x − coordinate of point>*
  ...
  <index of point >   *<x − coordinate of point>*
  end

  Elements:
  <indices of points of element>   *<classification (material-id)>*
  ...
  <indices of points of element>   *<classification (material-id)>*
  END

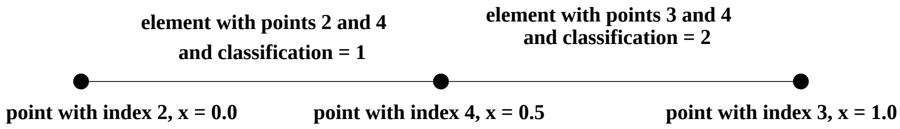

Figure 26: Partition of unit interval

Boundary:
<index of point on boundary> *<boundary conditon type>* *<classification>*
...
<index of point on boundary> *<boundary conditon type>* *<classification>*
END

The classification parameter has to be greater than 0 and less than 128. The same is valid in the 2d- and 3d-case.
As an example we present a unit interval divided into two intervals, see figure 26:

```
# 1D unit interval
Points:  maxIndex 4
2  0.0
4  0.5
3  1.0
end

Elements
2 4   1
3 4   2
end

Boundary
2 D 1
3 D 1
end
```

- 2D–region:

comments
Points: maxIndex *maxIndex*
<index of point > *<x,y – coordinates of point>*
...
<index of point > *<x,y – coordinates of point>*
end

Elements:
<indices of points of element> *<classification>*
...
<indices of points of element> *<classification>*
END

Boundary:
<indices of points of boundary edge> *<type>* *<classification>*
...
<indices of points of boundary edge> *<type>* *<classification>*
END

this section is optional:
Points:
<index of point on boundary> *<type>* *<classification>*
...
<index of point on boundary> *<type>* *<classification>*
END

In the section after the keyword `Boundary` you have to specify the boundary condition for each boundary edge. All the points on an edge inherit the boundary property from the edge, e.g. if a point belongs to a Neumann edge, it becomes a Neumann point itself. If it lies both on a Neumann edge and on a Dirichlet edge, it becomes a Dirichlet point. The keyword `Points` after the keyword `Boundary` opens the optional section for specifying boundary conditions on points. This special treatment of points may be necessary in some cases, e.g. if we have only Neumann edges and want to specify one point of Dirichlet type.

During the refinement process of our algorithm, a new point inherits the properties of the edge on which it lies. New edges or triangles inherit the properties of their fathers.

As an example we present a unit square divided into two triangles, see figure 27:

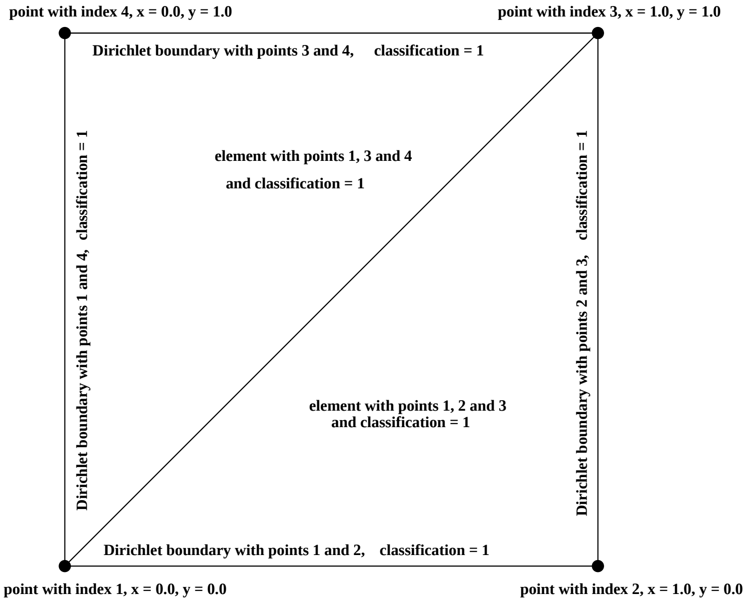

Figure 27: Partition of unit square

```
# 2D unit square
Points: maxIndex 4
1  0.0 0.0
2  1.0 0.0
3  1.0 1.0
4  0.0 1.0
end

Elements:
1 2 3  1
1 3 4  1
END

Boundary:
1 2  D 1
2 3  D 1
3 4  D 1
4 1  D 1
```

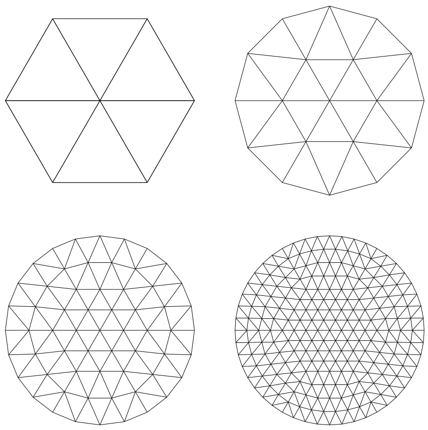

Figure 28: Approximation of a circular geometry by 3 uniform refinement steps

```
END
```

In the 2D–case we implemented a simple mechanism to approximate a circular geometry. The refinement algorithm computes new boundary points on the circle centered in the origin. After some steps of refinement the polygonal region is very similar to a circle.

As an example, look at figure 28.

This special handling of boundary edges is initiated by specifying an additional character for each curvilinear boundary edge, see for example the parameter file `circle.cmd` with the geometry description in `circle.geo` or `circle-2d.geo`.

```
# circle.geo

Points: maxIndex 10

1  1.0  0.0
2  0.5  0.866025404
3 -0.5  0.866025404
4 -1.0  0.0
5 -0.5 -0.866025404
6  0.5 -0.866025404
```

```
7  0.0  0.0
end

Elements:
1 2 7  1
2 3 7  1
3 4 7  1
4 5 7  1
5 6 7  1
6 1 7  1
END

Boundary:
1 2 d  1 s
2 3 d  1 s
3 4 d  1 s
4 5 d  1 s
5 6 d  1 s
6 1 d  1 s
END
```

The character `s` marks the edge for being approximating as circular arc during further refinement steps by placing the coordinates of a new point on this edge onto a circle centered in the origin ($x = 0, y = 0$) through the boundary points of the edge. Omitting the specification of the boundary shape or the character `P` provides the usual calculation of the coordinates as midpoint between the two boundary points of an edge.

Though our implementation of curvilinear boundaries is rather special (e.g. circular arcs are centered in the origin), it does not seem very difficult to generalize the concept.

As an example for a more complicated geometry we present the `.geo` file of example (9) among the static problems.

```
#  slit-2d-a.geo

Points: maxIndex 10
 1  -1.000000e+00   -1.000000e+00
 2   0.000000e+00   -1.414213562e+00
 3   1.000000e+00   -1.000000e+00
 4  -1.414213562e+00 0.000000e+00
 5   0.000000e+00    0.000000e+00
```

```
 6   1.414213562e+00 0.000000e+00
 7  -1.000000e+00    1.000000e+00
 8   0.000000e+00    1.414213562e+00
 9   1.000000e+00    1.000000e+00
 10  1.414213562e+00 0.000000e+00
END

Elements:
1 2 5  1
2 3 5  1
3 6 5  1
5 10 9 1
5 9 8  1
5 8 7  1
5 7 4  1
5 4 1  1
END

Boundary:
1  2 d 1 S
2  3 d 1 S
3  6 d 1 S
5  6 n 2
5 10 d 1
9 10 d 1 S
8  9 d 1 S
7  8 d 1 S
4  7 d 1 S
1  4 d 1 S
END


Points:
6 D 2
END
```

We present one more 2D–example showing how to define regions with holes. The domain is

$$\begin{aligned}\Omega &= \{(x,y)|0 <= x, y <= 10\}\\ &\setminus \{(x,y)|4 < x < 6 \wedge 2 < y < 4\}\\ &\setminus \{(x,y)|4 < x < 6 \wedge 6 < y < 8\}\end{aligned}$$

and one coarse triangulation is given by the following description, see also figure 28:

```
#  holes-2d.geo
Points: maxIndex 24
 1  0.0 0.0
 2  4.0 0.0
 3  6.0 0.0
 4  10.0 0.0
 5  0.0 2.0
 6  4.0 2.0
 7  6.0 2.0
 8  10.0 2.0
 9  0.0 4.0
 10  4.0 4.0
 11  6.0 4.0
 12 10.0 4.0
 13 0.0 6.0
 14 4.0 6.0
 15 6.0 6.0
 16 10.0 6.0
 17 0.0 8.0
 18 4.0 8.0
 19 6.0 8.0
 20 10.0 8.0
 21 0.0 10.0
 22 4.0 10.0
 23 6.0 10.0
 24 10.0 10.0
END

Elements:
2 1 5 1
2 6 5 1
2 3 6 1
7 3 6 1
4 3 7 1
4 8 7 1
5 9 6 1
6 10 9 1
7 8 11 1
```

```
8 12 11 1
9 10 13 1
10 14 13 1
10 11 14 1
11 15 14 1
11 12 15 1
12 16 15 1
13 14 17 1
14 17 18 1
15 19 16 1
16 20 19 1
17 21 18 1
18 21 22 1
18 19 22 1
19 22 23 1
19 20 23 1
20 23 24 1
END

Boundary:
1 2 d 1
2 3 d 1
3 4 d 1
4 8 d 1
8 12 d 1
12 16 d 1
16 20 d 1
20 24 d 1
24 23 d 1
23 22 d 1
22 21 d 1
21 17 d 1
17 13 d 1
13 9 d 1
9 5 d 1
5 1 d 1
6 7 d 2
7 11 d 2
11 10 d 2
10 6 d 2
14 15 d 3
15 19 d 3
19 18 d 3
```

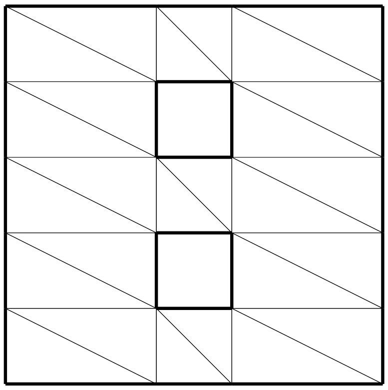

Figure 29: Triangulation of a quadratic geometry with two holes

```
18 14 d 3
END
```

- 3D–region:

  # comments
  Points: maxIndex *maxIndex*
  <index of point> *<x,y,z – coordinates of point>*
  ...
  <index of point> *<x,y,z – coordinates of point>*
  end

  Elements:
  <indices of points of element> *<classification>*
  ...
  <indices of points of element> *<classification>*
  END

  Boundary:
  <indices of points of boundary triangle> *<type>* *<classification>*
  ...
  <indices of points of boundary triangle> *<type>* *<classification>*

END

\# optional section:
Points:
<index of point on boundary> *<type>* *<classification>*
...
<index of point on boundary> *<type>* *<classification>*
END

In the section after the keyword `Boundary` you have to specify the boundary condition for each triangle. All the points on an triangle inherit the boundary property of the triangle. If a point belongs to a Neumann and to a Dirichlet triangle, it becomes a Dirichlet point. Edges are treated analogously.

The keyword `Points` after the keyword `Boundary` opens the section for specifying boundary conditions on points. This special treatment of points may be necessary in some cases, e.g.if we have only Neumann triangles and want to specify one point of Dirichlet type. This data section can be omitted.

During the refinement process of our algorithm, a new point inherits the properties of the edge on which it lies. New edges, triangles, or tetrahedra inherit the properties of their fathers.

As an example we present a unit cube divided into six tetrahedra:

```
#  unit cube
points: maxIndex  10
1  0 0 1
2  0 1 1
3  1 1 1
4  1 0 1
5  0 0 0
6  0 1 0
7  1 1 0
8  1 0 0
END

elements:
 1 2 3 6  1
 1 3 5 6  1
 3 5 6 7  1
 1 3 4 5  1
```

```
 4 5 7 8  1
 3 4 5 7  1
END

boundary:       # faces
1 2 3  d 1
1 2 6  d 1
1 5 6  d 1
2 3 6  d 1
3 6 7  d 1
5 6 7  d 1
1 3 4  d 1
1 4 5  d 1
4 5 8  d 1
4 7 8  d 1
3 4 7  d 1
5 7 8  d 1
END
```

We present as example for a little bit more complicated 3D–geometry a cube with a hole inside. The domain is

$$\begin{aligned} \Omega &= \{(x,y,z) | 0 <= x,y,z <= 1.0\} \\ &\setminus \{(x,y,z) | 0.25 < x,y,z < 0.75\} \end{aligned}$$

and a coarse tetrahedral mesh is given in `holes-3d.geo`, see also figure 30 showing a cut through $z = 0.5$. The faces outside are marked as Dirichlet type, the inner ones as Neumann type.

In `layer-3d.geo` you find the geometry similar to the last example, but the hole is filled with another material. The triangulation of the interior is generated by an additional point in $(0.5, 0.5, 0.5)$ and 48 tetrahedra. Figure 31 shows the configuration. On the boundary we defined Dirichlet conditions.

The generation of a sphere in 3D is analogous to the 2D-case. We also use the character `S` to mark a boundary face for curvilinear adaption during refinement. The ball must be centered in the origin. For example, see the files `circle3d.cmd`, `circle3d.geo`, `circle3d.mat`.

We can approximate a cylindrical region, too. On the initial grid we have to mark by a `C` those boundary faces which shall change during refinement to the curvilinear faces of a cylinder. For example, we start from the following cube (compare `cylindrical-3d-geo`)

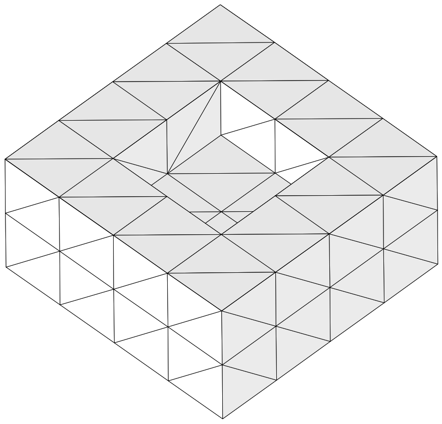

Figure 30: Cut through a cube with a hole

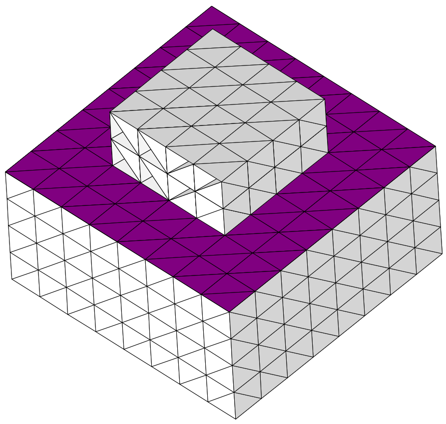

Figure 31: Cut through a cube consisting of two materials

```
#  cube with faces marked to approximate curvilinear faces
#  of a cylinder during the refinement process.

points: maxIndex  10
10 -1 -1 1
2  -1 1 1
3  1 1 1
4  1 -1 1
5  -1 -1 -1
6  -1 1 -1
7  1 1 -1
8  1 -1 -1
END

elements:
 10 2 3 6  1
 10 3 5 6  1
 3  5 6 7  1
 10 3 4 5  1
 4  5 7 8  1
 3  4 5 7  1
END

boundary:       # faces

10 2 3  d 1
10 2 6  d 1 C    # C or c  marks a curvilinear face
10 5 6  d 1 C
2  3 6  d 1 C
3  6 7  d 1 C
5  6 7  d 1
10 3 4  d 1
10 4 5  d 1 C
4  5 8  d 1 C
4  7 8  d 1 C
3  4 7  d 1 C
5  7 8  d 1
END
```

As an example, look at figure 32.

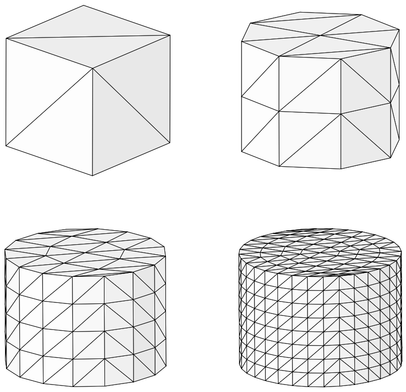

Figure 32: Approximation of a cylindrical geometry during 3 uniform refinement steps

In the future, we will realize a more general concept for curvilinear boundaries.

5.2 Ouput of Geometrical Data

We offer a set of formats to store the geometrical data of the actual refinement level into a file. The user can select one of the following formats.

- `writeZIBFormat`
  This format recommended by the ZIB is described in the preceding section.
- `writeGRAPEFormat`
  This format can be used by the graphical visualization software GRAPE.
- `writeAVSFormat`
  This format is evaluable by the graphical visualization software AVS.

The output is initiated by the command `writesolution=1`.

In additon to the geometrical data we write the values of the approximate solution in the nodes into the file. A file generated in the ZIB-format can be reread by the `file`-command. In this case the values in the nodes are ignored.

5.3 Material Coefficients, Boundary Conditions, and Initial Values

We select the coefficients (usually describing material properties) of the equation, the Neumann and the Cauchy boundary conditions by the command

```
material=***
```

where `***` represents one of the keywords in the left column of table 3.

Keyword	**materials**
`DefaultMaterial`	all coefficients are constant
`PeakSource`	coefficients in the differential operator are constant, the peak-shaped source function depends on spatial coordinates (e.g. used in example peak1d.cmd)
`MultiPeakSource`	similar to `PeakSource`, but a different source function
`TransPeakSource`	in transient problems: variable source function, all the other coefficients are constant
`StepSource`	in transient problems: all coefficients are piecewise constant (e.g. used in example u2step.cmd)
`StefanSource`	variable source function, all coefficients are piecewise constant (e.g. used in example stefan.cmd)
`UserStaticMaterial`	all coefficients and boundary conditions are variable and given by a function (e.g. used in example user-static-1d.cmd)
`UserTransientMaterial`	all coefficients and boundary conditions are variable and given by a function (e.g. used in example user-trans-1d.cmd)

Table 3: Keywords for material types

The boundary condition is chosen by the command

```
DirichletBCs=***
```

where `***` represents one of the keywords in the left column of table 4.

In general, the coefficients of the equation, the Dirichlet boundary values, and the initial values in transient problems are provided by member functions of the classes `Material` or `DirichletBCS` which are implemented in materials.cc, materialsA.cc, dirichlet.cc, dirichletA.cc,....

In the case of piecewise constant material coefficients or piecewise constant Dirichlet boundaries, it is not necessary to modify the source code. The user only has to set the

Keyword	dirichletBCs
`ConstDirichletBCs`	the solution is piecewise constant on the Dirichlet boundary
`UserDirichlet`	the Dirichlet boundary values are given by a function
`UserTransient`	the Dirichlet boundary and initial values are given by a function

Table 4: Keywords for Dirichlet boundary types

parameters `material=defaultMaterial` and `dirichletBCs=constDirichlet`. Then the description of these values is read from a `.mat`–file in which the material coefficients and the boundary conditions are specified like in the following example. The classifications of the region or boundary have to correspond to those of the geometry–file. If there is no `.mat`–file specified by the command

```
matFile=***
```

we suppose as default name the root name of the `.geo`-file but with the extension `.mat`.

This is a simple example for a mat–file:

```
Boundary:
    1 D 10.0
    1 N 0.0
    2 D 12.0
    2 N 1.0
    end

Materials:
    isotropic
    1 e 1 s 0 end
    2 e 2 s 1 end
    end
```

```
Factors:
    e 1e-4 end
```

The first integer in each row is the classification of the region, or boundary. It corresponds to the classification in the geometry file set for each element of the region or each face of the boundary. The classification parameter must be an integer greater than 0.

In our example we define values for boundary sections of classification 1 and 2. On sections with classification 1 or 2 and boundary type Dirichlet D, we have the constant values 10 and 12. On sections with boundary type Neumann N, we have homogeneous or inhomogeneous Neumann conditions.

On elements marked with 1, the elliptic coefficient is $k = 1$, and there is no source ($f = 0$) or mass term ($q = 0$).

On elements marked with 2, we have an elliptic coefficient $k = 2$, a constant source ($f = 1$), and no mass term ($q = 0$).

Material terms which are set to zero may be omitted.

If the keyword `Factors` occurs, then the values following after one of the keywords e (elleptic), s (source), m (mass), co (convection), c (Cauchy), n (Neumann), or d (Dirichlet) are used for scaling the corresponding values of the material or boundary defined in the `Materials` section before. We introduced this scaling to simplify the handling of physical units.

The next example (compare the preceding example 5) shows how to handle Cauchy boundary conditions. We have the unit square as geometry described in the file `unit-2d-A.geo`

```
# unit square

Points: maxIndex 4
1  0.0 0.0
2  1.0 0.0
3  1.0 1.0
4  0.0 1.0
end

Elements:
1 2 3  1
1 3 4  1
END
```

```
Boundary:
1 2  D 1        # Dirichlet
2 3  C 3        # Cauchy
3 4  N 2        # Neumann
4 1  D 1        # Dirichlet
END
```

The corresponding values of the boundary conditions are set in the `unit-2d-A.mat`–file.

```
Boundary:
1  D 10.0           # Dirichlet
2  N 0.0            # Neumann
3  C 1.0  2.0       # Cauchy
end

Materials:
isotropic
1  e 1  s 1  end
end
```

The Cauchy boundary condition is given by two values (compare (1) in appendix A), the first is α, the second is the right-hand side q_C. Here we have

$$\begin{aligned} u &= 10.0 && \text{on the boundary } x = 0 \text{ or } y = 0 \text{ (Dirichlet)} \\ u_y &= 0.0 && \text{on the boundary } y = 1 \text{ (Neumann)} \\ u_x + u &= 2.0 && \text{on the boundary } x = 1 \text{ (Cauchy)} \end{aligned}$$

The values specified in the `mat`–file are only relevant for the program in case of constant materials or constant boundary conditions. In one of the following sections we explain where to define nonconstant coefficients or boundary values in the source code.

6 System Solution and Preconditioning

6.1 The Iterative Solvers

We have implemented several iterative solvers. Anyone of these is invoked by the command

```
linSolver=***
```
,

where `***` represents one of the keywords in the left column of table 5.

Keyword	**Solver**
`cg`	conjugate gradient solver
`cgODir`	conjugate gradient solver with 3-term recurrence
`cr`	conjugate residual solver
`crODir`	conjugate residual solver with 3-term recurrence
`relax`	relaxation routine (Jacobi, SSOR etc.; for technical reasons the type is determined by the choice of a preconditioner applied to the Richardson iteration)
`gmRes`	GMRES of Saad and Schultz [SS86]
`stdBiCG`	bi-conjugate gradients ('standard version')
`biCGStab`	BiCGStab of van der Vorst [vdV92] ('stabilized version')
`cgs`	the CGS algorithm of Sonneveld [Son89]
`lsqCG`	conjugate gradients for the normal equations (CGNR)
`nonLinRelax`	nonlinear relaxation routine; to be used in conjunction with a nonlinear preconditioner

Table 5: Keywords for iterative solvers

In table 6 we list the commands which are relevant for the iterative solution procedures.

6.2 Preconditioners for Linear Problems

The possible choices are summarized in table 7.

6.3 Preconditioners for Nonlinear Problems

These types are summarized in table 8.

7 Error Estimation and Mesh Refinement

The error estimator provides estimates for the global and the local discretization errors. The global error is used to stop the algorithm when the requested precision is reached.

Parameter Name	**Default** (set in file kaskade.init)	**Description**
`directSolverLimit=`	`0`	limit for the direct sparse matrix solver on levels > 0 (see also `'level0direct'`)
`extPrecFactor=`	`1.0`	external precision factor; manipulates requested precision for convergence test in iterative solvers
`infoLinSystem=`	`25`	print information on every 25th iteration step
`level0direct=`	`3000`	limit for the direct sparse matrix solver on level 0; i.e. the solver will be used for a matrix dimension up to 3000
`linSolver=`	cg	determines the type of iterative solver
`maxOrthoGMRes=`	`12`	maximal number of vectors to be orthogonalized in GMRES
`omega=`	`1.0`	relaxation parameter for preconditioning with Jacobi- or SSOR- smoothing
`preConditioner=`	`MGsgs`	select one of the preconditioners
`timeLinSystem=`	`0 (false)`	print cpu-time for system solution

Table 6: Parameters for linear solvers and preconditioners

Keyword	Preconditioner
`Jacobi`	Jacobi type preconditioner
`SGS`	symmetric Gauss-Seidel preconditioner
`ILU`	preconditioning by incomplete LU-factorization
`MLJacobi`	multilevel preconditioner with Jacobi-type smoothing. The default relaxation parameter ω is 2/3
`MLSGS`	multilevel preconditioner with symmetric Gauss–Seidel smoothing. Forward Gauss–seidel is applied for pre-smoothing, the backward operation for post-smoothing
`AMLJacobi`	Additive multilevel preconditioner with Jacobi-type smoothing, also called BPX–preconditioner [BPX90]
`AMLSGS`	Additive multilevel preconditioner with Gauss–Seidel smoothing

Table 7: Preconditioners for linear problems

Keyword	Preconditioner
`nonlinSGGS`	single-grid Gauss-Seidel preconditioner
`nonlinMLGS`	multi-level Gauss-Seidel preconditioner
`trcNonlinMLGS`	truncated multi-level Gauss-Seidel preconditioner

Table 8: Preconditioners for nonlinear problems

The local information gives the error on each element on the actual mesh. We use it to determine a set of elements to be marked for refinement.

An estimator is selected by the parameter

```
errorEstimator=***
```

where `***` represents one of the keywords in the left column of table 9.

Keyword	**Estimator**
`DLY`	estimator due to [DLY89], see also [BEK92]
`modDLY`	modification of DLY (see [BEK93]), in case of constant coefficients and constant right-hand side identical with the standard DLY estimator,
`EFDLY`	3D–estimator due to [BEK93]
`RK1`	estimator for nonlinear problems due to [Kor95]
`RK2`	estimator for nonlinear problems, including one multigrid cycle for hierarchical defect correction, due to [Kor95]

Table 9: Keywords for Error Estimators

We offer some refinement strategies. The parameter

```
refStrategy=***
```

invokes one of these, where `***` represents one of the keywords in the left column of table 10.

Keyword	**Strategy**
`maxValue`	refinement with respect to the maximum of local errors
`extrapolation`	refinement with respect to extrapolated errors
`uniform`	uniform refinement

Table 10: Keywords for Refinement Strategies

The refinement strategy computes a threshold and marks those elements, where the estimated local error is beyond this threshold. Use the command

```
minRefRatio=***
```

to request a minimal number of elements to be refined. The default is 0.05, which means that at least 5 % of all elements will be refined.

If no estimator or no refinement strategy is selected, all elements will be refined.

8 How to Define a New Problem

In the preceding chapters we already learned how to solve a problem class of type (1) or (4) (see appendix A) with piecewise constant coefficients and piecewise constant boundary conditions. But a lot of applications also treat material properties varying in space, i.e. $k = k(x, ..)$, $q = q(x, ..)$ or $f = f(x, ..)$.

We summarize the steps for defining a new problem:

- Define the triangulation of the domain in a `.geo`–file
- Define boundary conditions and material properties (the coefficients of the equation) in a `.mat`–file, if piecewise constant, or in the file `userStatic.cc` resp. `userTrans.cc`, if variable.
- Specify your problem in a `.cmd`–file

8.1 Everything is variable

You can handle equations with variable coefficients and boundary conditions very easily by defining the coefficient functions in the file `userStatic.cc` for static problems (1) or in `userTransient.cc` for transient problems (4). In these files we prepared sample functions which can be overwritten by the user. They are named `E()` for the elliptic term `k`, `M()` for the mass term `q`, or `P()` for the coefficient of the time derivate in (4), and `S()` for the source term `f`. Each of these functions calls a subfunction depending on the space dimension. You only have to define the functions for the space dimension of your problem. Furthermore, it may be that you want to compare the approximate solution with a given function (e.g. for a simple test problem the known true solution). Such a function can be defined in the member function `trueSolInPoint()` and will be used if the parameter `compareSolution` is set to 1 in your `.cmd`-file.

In userStatic.cc we realized the following examples:

- 1–D:

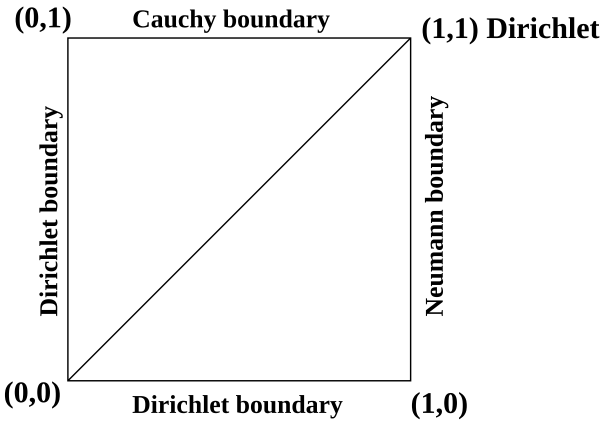

Figure 33: Geometry and boundary condition in example user_ static_2d

$$\begin{aligned} -\nabla x \nabla u + xu &= 1 - 4x + x^2(x-1) \qquad \text{in } \Omega \\ u &= x(x-1) \qquad \text{on } \Gamma_D \\ x\frac{\partial u}{\partial n} &= x(2x-1) \qquad \text{on } \Gamma_N \end{aligned}$$

- 2–D:

$$\begin{aligned} -\nabla xy \nabla u + (x+y)u &= x*y*(x+y) - x*x - y*y \qquad \text{in } \Omega \\ u &= xy \qquad \text{on } \Gamma_D \\ u &= 1 \qquad \text{in } (1,1) \\ xy\frac{\partial u}{\partial n} &= xy^2 \qquad \text{on } \Gamma_N \\ xy\frac{\partial u}{\partial n} + x*u &= 2x^2y \qquad \text{on } \Gamma_C \end{aligned}$$

Γ is defined in the file `user_static_2d.geo`, see figure 33 and the following description.

```
# unit square

Points: maxIndex 4
1  0.0 0.0
```

```
2  1.0 0.0
3  1.0 1.0
4  0.0 1.0
end

Elements:
1 2 3  1
1 3 4  1
END

Boundary:
1 2  D 1      # Dirichlet
2 3  N 1      # Neumann
3 4  C 2      # Cauchy
4 1  D 1      # Dirichlet
End

Points:
3 D 1         # Dirichlet
END
```

- 3–D:

$$\begin{aligned} -\nabla x \nabla u + (x+y+z)u &= xyz(x+y+z) - yz \qquad \text{in } \Omega \\ u &= xyz \qquad \text{on } \Gamma_D \\ x\frac{\partial u}{\partial n} &= xyz \qquad \text{on } \Gamma_N \end{aligned}$$

As an example, we give the definition of dummy functions for the elliptic coefficient k, the mass term q, and the source f:

```
Real UserStaticMaterial:: E(int type, Vector<Real>* x, int i, int j)
{
    Vector<Real>& xx = *x;

    switch(spaceDim)
    {
      case 1: return E1d(xx[1]);
      case 2: return E2d(xx[1], xx[2]);
      case 3: return E3d(xx[1], xx[2], xx[3]);
```

```
      default: abort(); return 0;
    }
}


Real UserStaticMaterial:: S(int type, Vector<Real>* x, Real time)
{
    Vector<Real>& xx = *x;

    switch(spaceDim)
    {
      case 1: return S1d(xx[1]);
      case 2: return S2d(xx[1], xx[2]);
      case 3: return S3d(xx[1], xx[2], xx[3]);
      default: abort(); return 0;
    }
}


Real UserStaticMaterial:: M(int type, Vector<Real>* x, int i, int j)
{
    Vector<Real>& xx = *x;

    switch(spaceDim)
    {
      case 1: return M1d(xx[1]);
      case 2: return M2d(xx[1], xx[2]);
      case 3: return M3d(xx[1], xx[2], xx[3]);
      default: abort(); return 0;
    }
}
```

The coefficient k is defined in the function `Real UserStaticMaterial::E` which calls one of the functions `E1d`, `E2d`, or `E3d` depending on the space dimension. The coefficient q is computed in the function M, and the source term f in the function S.

Available are functions for variable Neumann, Cauchy or Dirichlet boundary conditions. To specify a Cauchy condition we define two functions: the coefficient α of u is provided by the function `Real UserStaticMaterial::Cauchy(...)` and the right-hand side by the function `Real UserStaticMaterial::Neumann(...)`. In order to distinguish this Neumann part of the Cauchy boundary from other Neumann boundary faces, we have to use a common `class` identifier to the boundary face in the `.geo` file.

Though we make no use of it in our examples, we provided a function `Real UserStaticMaterial:: Inner(...)` which is only evaluated if the command parameter `innerBoundary` is set to 1. This is necessary if you want to compute an additional term of the form (3), appendix A. Please compare the examples (13) and (14) in chapter 3.

The point $(1,1)$ is declared to be of type Dirichlet in the `user-static-2d.geo` file. The values for this point and the other Dirichlet points are computed by the function `Real UserDirichletBCs::setBC(...)`.

For invoking your user problem of type (1) in appendix A, you just have to set the parameters

- `problem=staticHeat`
- `material=userStatic`
- `dirichletBCs=userStatic`

We provided `.cmd`-files with these commands:

`user-static-1d.cmd`, `user-static-2d.cmd`, and `user-static-3d.cmd`.

The `.geo`-files are named

`user-static-1d.geo, user-static-2d.geo, or user-static-3d.geo`.

Of course, you can choose other names.

In userTransient.cc you find the following 2–D problem:

$$\begin{aligned} \frac{\partial u}{\partial t} - \nabla \cdot \nabla u + u &= -2.0(x(x-1.0)+y(y-1.0))\ exp(-t) \qquad \text{in } \Omega \\ u &= x(x-1.0)y(y-1.0)\ exp(-t) \qquad \text{on } \Gamma_D \\ u(x,y,0) &= x(x-1.0)y(y-1.0) \qquad \text{in } \Omega \end{aligned}$$

A problem of type (4) in appendix A is invoked by the commands

- `problem=transientHeat`
- `material=userTransient`
- `dirichletBCs=userTransient`

You find sample `.cmd`–files setting these parameters:

`user-trans-1d.cmd`, `user-trans-2d.cmd`, and `user-trans-3d.cmd`.

The `.geo`–files are named `user-trans-1d.geo, user-trans-2d.geo`, or `user-trans-3d.geo`.

8.2 Something is variable, something is constant

In the last section we made the assumption that all the functions defining problem coefficients are variable. This allows an easy handling of the program, similar to the case of constant functions.

Of course you can mix problems of the form: constant Dirichlet boundary and variable material coefficients (including Neumann and Cauchy boundary), or vice versa. You have to select either

`dirichletBCS=ConstDirichlet` and `material=userStaticMaterial`

or `dirichletBCS=userDirichlet` and `material=DefaultMaterial`

It is a bit more complicated if some coefficients are piecewise constant and others are variable.

For example, we derived a material type `UserVarSource` for problems with piecewise constant coefficients k, q or c, and a variable right-hand side. This material can be selected by command

```
material=userVarSource
```

The source function f has to be specified in the class `UserVarSource` within file `materialA.cc`. The other coefficients, the Neumann or Cauchy boundary are expected to be constant with values defined in the `.mat`–file.

Other constellations of coefficients can be handled analogously by implementing a new `material` class.

9 Obtaining Run-Time Information

In the tables 11, 12, and 13 we list commands which are relevant for getting runtime information.

10 Technical Details for Programmers

10.1 Vector and Matrix Classes

We realized vector and matrix structures as special classes allowing index control during runtime. This is a very helpful feature during the development of new code, because the violation of array bounds is one of the most frequent errors. Of course,

Parameter Name	**Default** (set in file kaskade.init)	**Description**
`info=`	`0 (false)`	all information levels are activated or de-activated
`infoAb=`	`0 (false)`	general info about linear system
`infoErrorEstimator=`	`0 (false)`	info about error estimation
`infoLinSystem=`	`25`	print information on every 25th iteration step
`infoMG=`	`0 (false)`	info about multi-grid preconditioning
`infoPrecond=`	`0 (false)`	info about preconditioning
`infoRefinement=`	`1`	mesh: info levels ranging from 0 to 2, giving statistics about refinement steps
`infoSolution=`	`1 (true)`	info after each solution step
`printAb=`	`0 (false)`	print linear system
`printAL=`	`0 (false)`	print multi-level matrices
`printMatLabFormat=`	`0 (false)`	print stiffness matrix A in a format readable by MatLab
`printMesh=`	`0 (false)`	print triangulation (lists)
`printParameters=`	`0 (false)`	print command parameters

Table 11: Info– and print–commands

Parameter Name	**Default** (set in file kaskade.init)	**Description**
`graphic=`	`0 (false)`	switch graphic support on/off
`plotBoundary=`	`1 (true)`	plot no boundary when solution is plotted
`plotElements=`	`0 (false)`	plot no elements when the solution is plotted
`plotIsoLines=`	`1 (true)`	plot isolines of solution
`plotSolution=`	`1 (true)`	plot solution data after each space step
`plot3d=`	`0 (false)`	plot 2D–results as a surface plot (quasi–3d plot)
`plotKeep=`	`0 (false)`	keep all plots in separate windows
`plotLevels=`	`10`	desired number of iso-lines
`plotPointNodes=`	`0 (false)`	plot number of nodes at points
`plotSize=`	`0.4`	size of plot window
`plotTimeStep=`	`1`	plot the solution after each time step
`plotTriangleNodes=`	`0 (false)`	plot number of nodes at triangle
`postScript=`	`0 (false)`	generate postscript pictures of mesh and solution
`scaleX=`	`1`	scaling x-coordinate for quasi-3d plots
`scaleY=`	`1`	scaling y-coordinate for quasi-3d plots
`scaleZ=`	`1`	scaling z-coordinate for quasi-3d plots

Table 12: Plot–commands

Parameter Name	**Default** (set in file kaskade.init)	**Description**
`accTime=`	`0 (false)`	print all accumulated cpu times
`time=`	`0 (false)`	print cpu-time for all essential modules
`timeAssembler=`	`0 (false)`	print cpu-time for matrix assembling
`timeErrorEstimator=`	`0 (false)`	print cpu-time for error estimation
`timeLinSystem=`	`0 (false)`	print cpu-time for system solution
`timeRefinement=`	`0 (false)`	print cpu-time for mesh refinement
`timeTransport=`	`0 (false)`	print cpu-time for data transport between different grids
`timeUpdate=`	`0 (false)`	print cpu-time for update operation in node management

Table 13: Time information

such control is expensive in time. Therfore we provided a switch to turn on or off the array bound check by setting the flag `CheckBoundsFlag` in the files `vector.h` and `matrix.h` to `True` or `False` (default). A change of the flag is effective after recompiling the code.

10.2 Batchjobs

When the program fulfills a break condition (e.g the requested precision is reached) it stops with the prompt `REGULAR PROGRAM TERMINATION <CR>`. It terminates not before the user types the `RETURN`–key.

Sometimes, this pause is not very practical, e.g. if you want to run a sequence of batchjobs controlled by a shell script.

In such applications you can initiate an automatic termination of each job by setting the parameter `batchJob=1` in the corresponding `cmd`–file.

11 Graphical User Interface

Often it is difficult to learn the handling of a program and to understand the meaning of the parameters. To make these first steps easier, we support the user by a graphical user interface (`ZIBGui`). The `ZIBGui` is a manager only for the most important commands and parameters:

- It reads the usual `.cmd` files and displays the parameters. For practical reasons we changed the extension of those files to `.ex`.
- It changes parameters and generates a new parameter file.
- It starts the KASKADE executable.

The interface is based on the tool Tcl/Tk due to J. Ousterhout [Ouster94] for developing graphical interfaces. It is independent of the numerical code KASKADE . Each `.ex` file is recognized as an example and the parameters in it can be interpreted and changed. However, the use of the parameters is more restrictive than for the parser in the KASKADE program. Only those parameters are understood correctly which concur fully with the notation of the table in appendix B. Abbreviations are not allowed, capital letters are significant.

You find a detailed description (in German) in the manuals [NPR96] and [NPRW96] including examples for customizing the user interface for your changes in the numerical application, e.g. you have a new parameter.

11.1 How to install the ZIBGui environment

First install the public domain software Tcl/Tk (version `Tcl 7.4` and `Tk4.0`) on your system. Then execute the following steps in order to translate the `ZIBGui` sources which you find in subdirectory `/zgui`:

- Goto subdirectory `/zgui`.
- Set in the `Makefile` the path to the directories of your Tcl/Tk libraries and include files. Additionally adapt the name of your operating system, for instance `OS = SUNOS`. This name will be used to store object-files and libraries in the subdirectories `obj-OS`*name* and `lib-OS`*name*.
- Type `make` to compile and link the files. The name of the generated executable is `zgui`.
- Set in the file `zgui/k6.tcl` the path to the directory with the KASKADE application `k6`. In the standard case the path is the parent directory of the `zgui`

Figure 34: Starting menu of the graphical user interface

directory. Further you can modify the path to the library with the tcl-files of `zgui`. By default it is `zgui/library`.

- If you have not already installed the KASKADE executible `k6`, please do it now.
- Simply start the graphical interface by typing

```
> zgui
```

11.2 Examples

When `ZIBGui` is started, it opens a window similar to that shown in figure 34. (Please note: If the following pictures of some `ZIBGui` windows are a little bit blurred, try to print these pages on a 600dpi printer.)

The menus of the graphical user interface (ZIBGui) reflect the structure of the numerical software. Before starting the KASKADE code by clicking on the `Start`-button in the `application control` board you may change your mathematical problem (type, material, and boundary conditions) , the components of the multilevel solver (linear solver, preconditioner, convergence criteria, error estimator), and select some graphical and protocol parameters in the submenus of the `Settings` menu in the root window.

There are default values which configure a simple static heat conduction problem on the unit square in 2 space dimensions.

Meanwhile we added a `Postscript`-button to the `application control` board which can be used to generate postscript pictures of the actual mesh and approximate solution.

In the root window you find the `Example`–menu. It offers a set of prepared examples. Each example is a collection of parameter settings stored in a file (extension `.ex`). To create your own example file you can use the `saveAs` or `save` field in this menu. By `save` you can overwrite only the example you selected before. Maybe, you have overwritten an example but want to reestablish the example settings recommended by the authors, then you can select `Default` in the `Example`–menu.

It may be that for a special problem type the selection of some other parameters (materials, boundary conditions, or numerical methods) makes no sense. In this case these widgets should be disabled.

Changes in one of the settings windows have to be confirmed with the `Apply`–button before being valid for the next run. The `Default`–button resets all options to the default values you get when you start the program. By the `Reset`–button the state before the last `Apply`–click is reestablished.

A path through the program:

- Select the type of your problem. In the user interface we use the name `staticHeat` synonymous to linear, scalar elliptic problem, and the name `transHeat` to linear, scalar parabolic problem, because we cannot solve general elliptic or parabolic equations.

- Use the `geo-file` button to select the `.geo` file which describes the triangulation (1D, 2D, or 3D) of the region Ω.

- The `.mat-file` button offers the possibility to select a `.mat` file describing piecewise constant material coefficients or constant boundary conditions. If you have variable coefficients and boundary conditions, the specification of a `.mat` file is not necessary.

- The global precision parameter specifies the error tolerance for the solution process. The adaptive process will be continued until the estimated error lies under this threshold. The value for maximal level provides a possibility to stop the multilevel solver after a certain number of steps independently of the achieved precision.

- Now you can choose the setup for the multilevel solver. First you may select the iterative method to compute the solution of linear systems. The system may be preconditioned by one of the offered methods. Depending on the problem type you have some suitable error estimators which can provide an estimation of the global error. This is important to control the multilevel process. In addition the estimator computes local errors which can be used together with a refinement strategy to generate an adaptive grid.

- The coupling of requested precision in the iterative solver of the linear system and the estimated discretization error is tuned by the convergence test strategy. Both the `residual`– and the `cascadic iteration`–method are used for elliptic problems. In general the former one is recommended, see [BER95].
- For parabolic equations you find entries in the `problem definition` window for the `start time` and `final time`.
- The `graphics` setting menu provides different graphical presentations of the solution process, e.g. plots of the solution and the grid. The graphic output is X11–based.
- The amount of information of the program during runtime is tuned by the `protocol` settings. For further evaluations of the protocol information we introduce the `Special Statistic` button. The tool which uses this additional output is not yet part of the KASKADE package.

For 3–D examples, there is no interactive graphic support integrated in this version of the program.

Some examples:

- **default:**

  This is the example which is automatically selected when the program starts.

  - **Problem**

    Static heat conduction in two space dimensions, constant coefficients $k = 1$, and constant right–hand side $s = 1$, and homogeneous boundary conditions.

    Problem `settings`:

    * `Type` = staticHeat, `Material` = constant, `Dirichlet` = constant
    * `geo-file` selection: file = unit–2d.geo (unit square)
    * `mat-file` selection: file = unit–2d.mat
    * $\cdots$

  - **Numerics**

    Conjugate gradient method with a multigrid preconditioner (sym. Gauss–Seidel smoother), error estimation (Deuflhard, Leinen, Yserentant, 1989, see [DLY89]) and adaptive refinement.

    Numerical parameters:

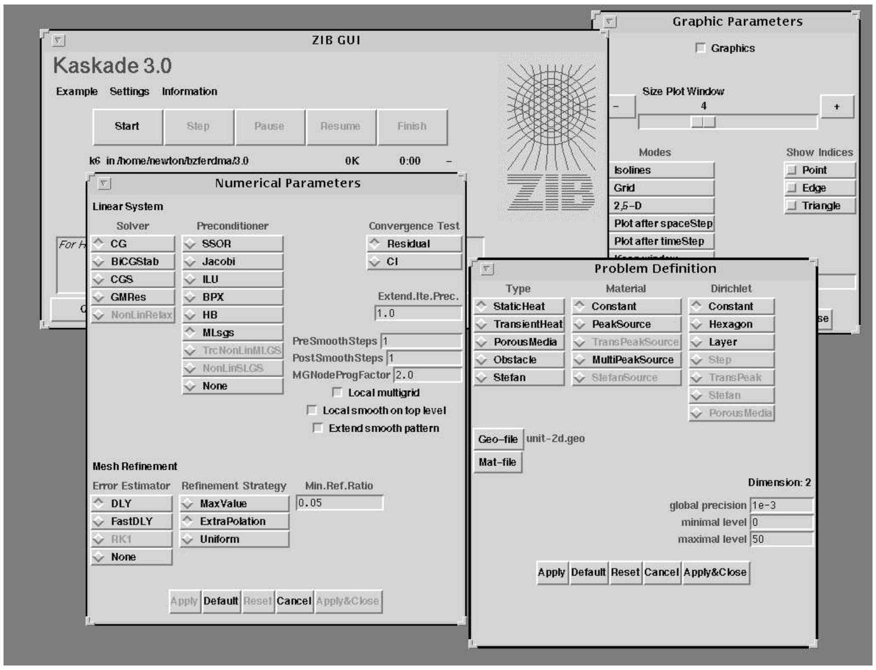

Figure 35: Menus for setting the problem parameters

* `linear solver` = cg
* `preconditioner` = MLsgs
* `error estimator` = DLY
* `refinement strategy` = extrapolation
* ⋯

Variation of parameters:

– change the region by the `geo-file` button, e.g. select unit–1d.geo, unit–3d.geo, hexagon.geo, l–2d.geo, l–3d.geo, slit–2d.geo, or corner.geo.

  The space dimension of the problem is automatically provided by analyzing the the `.geo`–file. Then the program selects the appropriate numerical algorithms. Of course, the material data must be reasonable for the geometry, e.g. if the region includes a Neumann boundary, the values of it must be defined in the corresponding `mat`–file (if values are piecewise constant) or in a member function of the selected material (if values are not piecewise constant). We recommend to use always a `.mat` file with the same root name as the `.geo` file.

- change the coefficients in the equation, e.g. select `Material=PeakSource` (peak in the source term) or `Material=MultiPeakSource`.
- change the numerical parameters in the `Numerics` window, e.g. select different linear solvers and combine them with some preconditioners, or test different refinement strategies. For example, a computation without error estimation (`error estimator` = `None`) or with uniform refinement (`refinement strategy` = `uniform`) is quite interesting, if you have selected the problem with a peak source (`Material` = `peakSource`). In the uniform case you need 5 refinement steps and about 2000 nodes to achieve an accuracy of 2%. In the adaptive case you need 5 steps, too. But less than 500 nodes. This example shows the efficiency of adaptive methodes.

- **boundaryLayer**

  The solution of this elliptic 2D–problem has a boundary layer along the sides $x = 1$ and $y = 1$.

  - **Problem**

    equation: $\Delta u - 100u = 0$

    domain: unit square

    boundary condition: Dirichlet defined by $\frac{\cosh(10x)+\cosh(10y)}{2\cosh(10)}$
  - **Numerics**

    In this example, it is not reasonable to vary the material data, but the user can test the different numerical parameters like in the default example.

- **corner**

  The solution of this Laplace equation on a reentrant corner 2D–problem has a singularity in the origin.

  - **Problem**

    equation: $\Delta u = 0$

    domain: part of 2D–hexagon with reentrant corner

    boundary condition: On parts of the boundary we have homogeneous Neumann conditions which are defined in the file `corner.mat`.
    On the rest of the boundary there are non-constant Dirichlet conditions $u(x,y) = r^{1/2}$, with $r = \sqrt{x^2 + y^2}$.

    They are defined in the source file dirichletA.cc. In near future, we will present a function in the ZIBGui in order to modify interactively parts of the code and to recompile the executable automatically.
  - **Numerics**

    In this example the user can test the different numerical parameters like in the default example.

- **staticPeak**

  The solution of this stationary Poisson problem has a peak at a point.

  - **Problem**

    equation: $-\Delta u = f$, the source f has a peak of the form $f \sim c \cdot e^{-p(x-a)^2}$ in the 1D-case, and analogously in higher dimensions. It is defined in the source file `materialsA.cc` as a function of the coordinates.

    domain: unit interval in 1D, unit square in 2D, cube in 3D.

    boundary condition: homogeneous Dirichlet

  - **Numerics**

    In this example, it is not reasonable to vary the material data or boundary conditions, but the user can test the different numerical parameters like in the default example. It is very interesting to study the differences between adaptive and uniform refinement.

  Variation of parameters:

  - change the region by the `geo`–file button, e.g. select peak-2d.geo, peak-3d.geo.
  - change the numerical methods similarly to the tests in the default example.

- **transientPeak**

  The solution of this parabolic problem has a peak at a point of the domain.

  - **Problem**

    equation: $u_t - \Delta u = f$, the source f has a time–dependent peak of the form $f \sim c \cdot e^{-p(x-a(t))^2}$ in the 1D-case, and analogously in higher dimensions. It is defined in the source file `materialstr.cc` as function of the coordinates and the time.

    domain: unit interval in 1D, unit square in 2D, unit cube in 3D.

    boundary condition: homogeneous Dirichlet conditions.

  - **Numerics**

    In this example, it is not reasonable to vary the material data or boundary conditions, but the user can test the different numerical parameters like in the default example.

  Variation of parameters:

  - Change the region by the `geo-file` button, e.g. select transpeak–2d.geo, unit-3d.geo. We recommend to request less accuracy (factor `globalPrecision`) in higher space dimensions.

  – Change the numerical methods like in the default example.

- **transientDiffusion**

  The solution of this parabolic problem shows a diffusion process starting from a sharp initial profile.

  – **Problem**

  equation:

$$1000u_t - \Delta u = \begin{cases} 1.0, & \text{for } t \leq 10.0 \\ 0.0, & \text{for } t > 10.0 \end{cases}$$

  The coefficients are described in the file `u-1d-step.mat`, `u-2d-step.mat`, resp. `u-3d-step.mat` depending on the space dimension.

  domain: unit interval in 1D, unit square in 2D, cube in 3D.

  boundary condition: u=0 on the left side, u=1 opposite to it, and homogeneous Neumann condition on the other parts of the boundary.

  initial value: is given as function in the file `dirichlettr.cc` and is available under the name `Step`.

  – **Numerics**

  This example demonstrates the parabolic solver, which is adaptive in space and time. In each time step, the computation of the spatial solution starts from the initial grid.

  In this example, it is not reasonable to vary the material data or boundary conditions, but the user can test the different numerical parameters like in the default example.

  Variation of parameters:
  change the region by the `geo-file` button, e.g. select u-1d-step.geo, u-2d-step.geo, or u-3d-step.geo. If the user changes the geometry in this example, he has also to select the `mat-file` u-1d-step.mat, u-2d-step.mat, or u-3d-step.mat with the suitable material and boundary condition.

- **obstacle**

  Here we present a special stationary obstacle problem. In this class of problems we have to solve variational inequalities with constraints. In chapter 3, nonlinear problem (1), we give a more detailed description of this 2D–problem.

- **stefan**

  The stefan problem describes the transition between two phases of a material, e.g. a melting iceberg.

- **Problem**

  This time-dependent problem is formulated in form of variational inequalities. See chapter 3, nonlinear problem (2) for a detailed description of this 2D–problem.

- **Numerics**

  For this class of nonlinear problems we provide a special error estimator and two suitable multigrid methods as linear solver.

- **porousMedia**

  This example shows the time-dependent diffusion in porous media. In chapter 3, nonlinear problem (3), you find more informations on this 2D–problem.

12 Frequently Asked Questions

Here we present a list of frequently asked questions

- **Which** file is responsible of what?
- **How** to add a convective term?
- **How** can we handle periodic boundary conditions?

Shortly you can read here the answers, too.

A Problem Classes

Here we give a description of the installed problem classes.

Static Heat Conduction and Related Problems

This problem class involves elliptic partial differential equations of the type

$$\begin{aligned} -\nabla k \nabla u + qu &= f && \text{in } \Omega \\ u &= u_0 && \text{on } \Gamma_D \\ k\frac{\partial u}{\partial n} &= q_N && \text{on } \Gamma_N \\ k\frac{\partial u}{\partial n} + \alpha\, u &= q_C && \text{on } \Gamma_C \end{aligned} \tag{1}$$

where k and α denote material parameters (thermal conductivity and transfer coefficient). Ω may be a one-, two-, or three-dimensional region; boundary conditions of Dirichlet, Neumann, or Cauchy type are applied on the corresponding boundary sections Γ_D, Γ_N, and Γ_C. The source f and the coefficients k, q, and α may depend on space coordinates. An extension to anisotropic materials with tensors $k = k_{ij}$ is quite straightforward.

The weak formulation associated with (1) is: Find $\hat{u} \in H^1_{\Gamma_D}(\Omega)$ satisfying

$$a(\hat{u}, v) = G(v) \qquad \forall v \in H^1_0(\Omega) \tag{2}$$

where

$$\begin{aligned} a(u,v) &= \int_\Omega (k \nabla u \nabla v + quv) dxdydz + \int_{\Gamma_C} \alpha uvds \\ G(v) &= \int_\Omega fvdxdydz + \int_{\Gamma_C} q_C vds + \int_{\Gamma_N} q_N vds \end{aligned}$$

There is the possibility in KASKADE to compute one additional boundary integral on the right-hand side of the equation 2, e.g. due to a special source in the region. It has the form:

$$G(v) = \ldots + \int_{\Gamma_I} q_I vds \tag{3}$$

Here Γ_I denotes an interior layer, e.g. the face between two different materials, and q_I is a function defined on this layer.

Transient Heat Conduction

This class comprises linear parabolic problems of the form

$$c\,\frac{\partial u}{\partial t} - \nabla k \nabla u + qu \;=\; f \qquad \text{in } \Omega \tag{4}$$

The boundary conditions are like in (1), but may be time-dependent like the source term f. Additionally we have to define initial values for $t = 0$:

$$u(x,0) \;=\; u_0(x) \qquad \text{in } \Omega$$

$u_0(x)$ has to be supplied by the user or can be calculated via the solution of a static heat conduction problem.

Nonlinear Problems

Here, the problem classes have an additional function describing the nonlinearity. It defines the nonlinear characteristics of the problems, e.g. obstacle functions or a nonlinear heat capacitance.

Obstacle Problems

Two obstacle functions $\varphi_{up}(x)$, $\varphi_{low}(x)$ can be added to a differential equation of type (1):

$$\begin{array}{rcl} u(x) & < & \varphi_{up}(x) \\ u(x) & > & \varphi_{low}(x) \end{array} \tag{5}$$

Such constraints may occur in various types of applications. The corresponding variational inequality yields a nonlinear system, which is solved by a single-level Gauss-Seidel or a monotone multigrid method [Kor95].

Stefan Problems and Related Nonlinear Equations

These problems may be regarded as degenerate parabolic initial value problems of the form

$$\begin{array}{rcll} \frac{\partial}{\partial t} H(u) - \nabla k \nabla u & = & f & \text{in } \Omega \\ u(x,0) & = & u_0(x) & \text{in } \Omega \\ H(x,0) & = & H_0(x) & \text{in } \Omega \end{array} \tag{6}$$

combined with boundary conditions like in (1). H is the heat content or a generalized enthalpy. In KASKADE H is the discontinuous derivative of a continuous piecewise quadratic function of u. Thus H must be specified as a piecewise linear function of u; a discontinuity in H defines a change of phase with a certain amount of latent heat.

Other nonlinear problems may be formulated in a similar way, e.g. the porous media equation (see [Kor95]). Of course, we may have additional obstacle functions of the form (5).

B Command Listing

On the following pages we present an alphabetical list of the command parameters:

Parameter Name	**Default** (set in file kaskade.init)	**Description**
`accTime=`	`0 (false)`	print all accumulated cpu times
`absPrecision=`	`0 (false)`	see parameter `globalPrecision`
`batchJob=`	`0 (false)`	the program terminates automatically when fulfilling a break condition
`cmd=`		input command file
`compareSolution =`	`0 (false)`	compare approximate solution with the true solution in the grid nodes
`cycle=`	`0 (false)`	new boundary points are placed on a circle (2d) or ball (3d) around the origin
`cylinder=`	`0 (false)`	new boundary points are placed on a circle around the origin parallel to the plane z=0
`directSolverLimit=`	`0`	limit for the direct sparse matrix solver on levels > 0 (see also `'level0direct'`)
`dirichletBCs=`	`constDirichlet`	identifies the type of Dirichlet boundary condition
`errorEstimator=`	`DLY`	select an error estimator
`extPrecFactor=`	`1.0`	external precision factor; manipulates requested precision for convergence test in iterative solvers
`file=`	`unit-2d.geo`	name of geometry file (the extension `.geo`) need not be specified. The material file is expected to have the same name, but the extension `.mat`.
`globalPrecision=`	`1e-3`	desired relative precision (maximum discretization error with respect to global energy) if `absPrecision`=0, otherwise desired absolute precision
`graphic=`	`0 (false)`	switch graphic support on/off
`info=`	`0 (false)`	all information levels are activated or deactivated
`infoAb=`	`0 (false)`	general info about linear system

Parameter Name	Default (set in file kaskade.init)	Description
`infoErrorEstimator=`	`0 (false)`	info about error estimation
`infoLinSystem=`	`25`	print information on every 25th iteration step
`infoMG=`	`0 (false)`	info about multi-grid preconditioning
`infoPrecond=`	`0 (false)`	info about preconditioning
`infoRefinement=`	`1`	mesh: info levels ranging from 0 to 2, giving statistics about refinement steps
`infoSolution=`	`1 (true)`	info after each solution step
`level0direct=`	`3000`	limit for the direct sparse matrix solver on level 0; i.e. the solver will be used for a matrix dimension up to 3000
`linSolver=`	cg	determines the type of iterative solver
`localExtend=`	`1 (true)`	extend smoothing pattern (=0: only new nodes, =1: new nodes + neighbours)
`localOnTop=`	`1 (true)`	local smoothing of top-level matrix
`localSmooth=`	`1 (true)`	local multigrid
`material=`	`defaultMaterial`	name identifying the material type, `defaultMaterial` defines constant coefficients on the elements
`maxOrthoGMRes=`	`12`	maximal number of vectors to be orthogonalized in GMRES
`maxRefSteps=`	`50`	maximal number of adaptive refinement steps in solve
`minRefRatio=`	`0.05`	elements marked for refinement: at least 5%
`minRefSteps=`	`0`	minimal number of adaptive refinement steps in solve
`nPostSmooth=`	`1`	number of post-smoothing steps in multigrid
`nPreSmooth=`	`1`	number of pre-smoothing steps in multigrid
`omega=`	`1.0`	relaxation parameter for preconditioning with Jacobi- or SSOR- smoothing

Parameter Name	**Default** (set in file kaskade.init)	**Description**
`pause=`	`1 (true)`	stop when function `Pause()` is called in the code and wait for input: `<CarriageReturn>` → continue until next `Pause()` is encountered, `'c'` → continue and disable the function `Pause()`, `'p'` → generate a picture in postscript format of the actual mesh and the approximate solution, `'q'` → quit program
`plotBoundary=`	`1 (true)`	plot no boundary when solution is plotted
`plotElements=`	`0 (false)`	plot no elements when the solution is plotted
`plotSolution=`	`1 (true)`	plot solution data after each space step
`plot3d=`	`0 (false)`	plot 2D–results as a surface plot (quasi–3d plot)
`plotIsoLines=`	`1 (true)`	plot isolines of solution
`plotKeep=`	`0 (false)`	keep all plots in separate windows
`plotLevels=`	`10`	desired number of iso-lines
`plotPointNodes=`	`0 (false)`	plot number of nodes at points
`plotSize=`	`0.4`	size of plot window
`plotTimeStep=`	`1`	plot the solution after each time step
`plotTriangleNodes=`	`0 (false)`	plot number of nodes at triangle
`postScript=`	`0 (false)`	generate postscript pictures of mesh and solution
`postTimeStep=`	`0`	generate postscript pictures of mesh and solution for timestep which fulfills timestep%postTimeStep==0 and timestep > 0
`preConditioner=`	`MGsgs`	select one of the preconditioners
`printAb=`	`0 (false)`	print linear system
`printAL=`	`0 (false)`	print multi-level matrices
`printMatLabFormat=`	`0 (false)`	print stiffness matrix A in a format readable by MatLab
`printMesh=`	`0 (false)`	print triangulation (lists)
`printParameters=`	`0 (false)`	print command parameters
`problem=`	`staticHeat`	specifies the problem type to be allocated and solved

Parameter Name	**Default** (set in file kaskade.init)	**Description**
`readOldFormat=`	`0 (false)`	read the geometry input in old ZIB–format
`readZIBFormat=`	`1 (true)`	read the geometry input in new ZIB–format
`refStrategy=`	`extrapolation`	select the refinement strategy
`refTetrahedron=`	`shortestEdge`	method for refinement of a tetrahedron
`scaleX=`	`1`	scaling x-coordinate for quasi-3d plots
`scaleY=`	`1`	scaling y-coordinate for quasi-3d plots
`scaleZ=`	`1`	scaling z-coordinate for quasi-3d plots
`spaceDim=`	`2`	space dimension
`time=`	`0 (false)`	print cpu-time for all essential modules
`timeAssembler=`	`0 (false)`	print cpu-time for matrix assembling
`timeErrorEstimator=`	`0 (false)`	print cpu-time for error estimation
`timeLinSystem=`	`0 (false)`	print cpu-time for system solution
`timeRefinement=`	`0 (false)`	print cpu-time for mesh refinement
`timeTransport=`	`0 (false)`	print cpu-time for data transport between different grids
`timeUpdate=`	`0 (false)`	print cpu-time for update operation in node management
`writeAVSFormat=`	`0 (false)`	set format for geometrical output, it corresponds to AVS visualization tools, only for 3D objects
`writeCpuTime=`	`0 (false)`	write cpu–time information to info–file
`writeGrapeFormat=`	`0 (false)`	set format for geometrical output, it corresponds to GRAPE visualization tools, only for 3D objects
`writeIterations=`	`0 (false)`	write iteration numbers to info-file
`writeZIBFormat=`	`0 (false)`	set format for geometrical output, it corresponds to the standard ZIB–format
`writeSolution=`	`0 (false)`	allow output on a file, the format of output has to be defined by one of the commands writeZIBFormat, writeGRAPEFormat, writeAVSFormat
`writeTimeStep=`	`0 (false)`	write the solution after each time step

References

[Bor90] F.A. Bornemann. *An Adaptive Multilevel Approach to Parabolic Equations I, II.* IMPACT Comput. Sci. Engrg. 2, 1990, and IMPACT Comput. Sci. Engrg. 3, 1991.

[BEK92] F. Bornemann, B. Erdmann, R. Kornhuber. *Adaptive Multilevel–Methods in Three Space Dimensions.* Int. J. Numer. Meths. in Eng., 36, 1993.

[BEK93] F. Bornemann, B. Erdmann, R. Kornhuber. *A Posteriori Error Estimates for Elliptic Problems in Two and Three Dimensions.* Preprint SC 93-29, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1993.

[BER95] R. Beck, B. Erdmann, and R. Roitzsch. *KASKADE 3.0: An Object-Oriented Adaptive Finite Element Codes.* Technical Report TR 95-4, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1995.

[BPX90] J.H. Bramble, J.E. Pasciak and J. Xhu. *Parallel Multilevel Preconditioners.* Math. Comp, Vol 55, 1990.

[Cr88] J. Crank. *Free and Moving Boundary Problems..* Oxford University Press, Oxford, 1988.

[DLY89] P. Deuflhard, P. Leinen, and H. Yserentant. *Concepts of an Adaptive Hierarchical Finite Element Code.* IMPACT Comput. Sci. Engrg. 1, 1989.

[EO82] C. M Elliot, J. R. Ockendon. *Weak and Variational Methods for Moving Boundary Problems, Reseach Notes in Mathematics 53.* Pitman, London, 1982.

[GRA] GRAPE, Graphical Programming Environment. Sonderforschungsbereich 256. University of Bonn, Germany.

[HS90] K. H. Hoffmann and J. Sprekels, editors. *Free Boundary Value Problems.* Birkhäuser, Basel, 1990.

[HYP89] P. Wust, J. Nadobny, R. Felix, P. Deuflhard, W. John, A. Louis. *Numerical approaches to treatment planning in deep RF-Hyperthermia.* Strahlenther. Onkol. 165, p. 751-757, 1989.

[Kor95] R. Kornhuber. *Monotone Multigrid Methods for Nonlinear Variational Problems.* Habilitationsschrift, Freie Universität Berlin, Berlin, 1995.

[Mus37] M. Muskat. *The Flow of Homogeneous Fluids Through Porous Media..* McGraw–Hill, New York, 1937.

[NPR96] U. Nowak, U. Pöhle, R. Roitzsch. *Eine graphische Oberfläche für numerische Programme.* Internal report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1996. Available as postscript file (`zgui-german.ps.Z` in subdirectory pub/kaskade/Manuals/3.x) by anonymous ftp to machine elib.zib-berlin.de.

[NPRW96] U. Nowak, U. Pöhle, R. Roitzsch, R. Werk. *ZGUI – Manual.* Internal report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1996.

[Ouster94] J. Ousterhout. *Tcl and the Tk Tollkit.* Addison-WesleyPublishing Company, Professional Computing Series, 1994.

[Pen48] H. H. Pennes. *Analysis of tissue and arterial blood temperatures in the resting human forearm.* J. Appl. Physiol. 1, p. 93-122, 1948.

[Rodr87] J. F. Rodrigues. *Obstacle Problems in Mathematical Physics.* Mathematical Studies 134, North Holland, Amsterdam, 1987.

[Roi93] B. Erdmann, J. Lang, and R. Roitzsch. *KASKADE Manual Version 2.0: FEM for 2 and 3 Space Dimensions.* Technical Report TR 93-5, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1993.

[See90] M. Seebaß. *3D-Computersimulation der interstitiellen Mikrowellen-Hyperthermie von Hirntumoren.* Bericht Nr. 1/90, Inst. f. Radiologie und Pathophysiologie, Deutsches Krebsforschungszentrum Heidelberg, 1990.

[Son89] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Stat. Comp.*, 10(1), 1989.

[SS86] Y. Saad and M. H. Schultz. *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems.* SIAM J. Sci. Stat. Comp., 7(3), July 1986.

[vdV92] Henk A. van der Vorst. *BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems.* SIAM J. Sci. Stat. Comp., 13(3), March 1992.