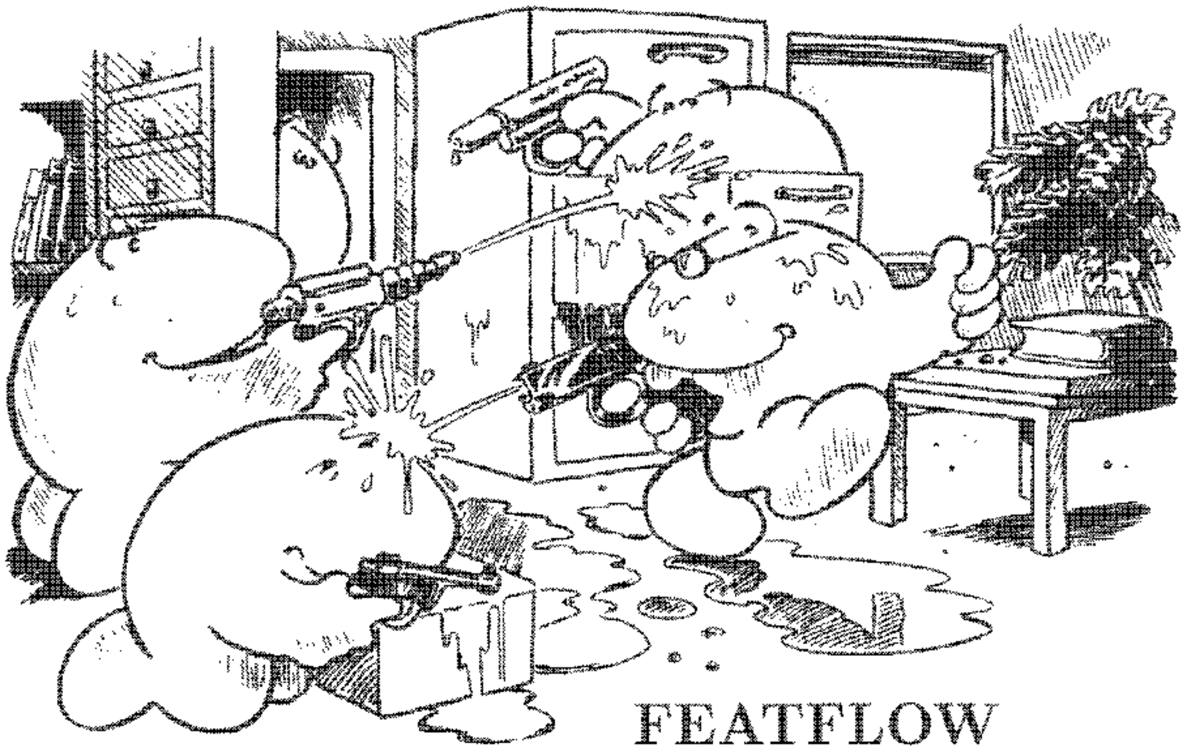

FEATFLOW

Finite element software for the incompressible Navier–Stokes equations

User Manual
Release 1.1

S. Turek, Chr. Becker
University of Heidelberg, Institute for Applied Mathematics
Im Neuenheimer Feld 294, D–69120 Heidelberg, Germany

Heidelberg, 02/01/1998

Contents

Short description

The progam package FEATFLOW is both a user oriented as well as a general purpose subroutine system for the numerical solution of the incompressible Navier–Stokes equations in two and three space dimensions. FEATFLOW is part of the FEAST project which has the aim to develop software which realizes our new mathematical and algorithmic ideas in combination with high performance computational techniques. For more information about this project ask the authors or look at the Internet–URL:

http://www.iwr.uni-heidelberg.de/~featflow

which contains most of our research activities, including the 'Virtual Album of Fluid Motion'. FEATFLOW is designed for the following three classes of applications:

Education:

Students coming from Mathematics, Physics, Engineering sciences or Computer sciences learn to handle modern numerical software. This helps for future work in industry or for doing Diploma or Ph.D. works since the basics of today's numerical software engineering, concerning software and hardware, are provided. Actually, we realize this education program as 'Software–Praktikum' at the University of Heidelberg.

Research:

This software (and hence the underlying Mathematics) is running at several universities in Europe and North–America. As a basic tool it helps to verify own codes by comparisons with reference configurations, and it can be used for adding new components to solve more complex problems. For instance, in our group it is the basic solver which is used, after slight modifications, for exploring new algorithms for compressible media, turbulence modelling, multi–phase and non–newtonian flows or shape optimization.

Industry:

Since this software is purely based on a mathematically optimized approach it should be able (if everything is implemented in a correct way) to be much more efficient than most actual software tools used by industry. Therefore, we try not only to solve "mathematical" test problems, but realistic applications, too. We hope to demonstrate, that this approach is not only faster and more efficient than most other programs, but also applicable for the same problems. Therefore, one of our main goals is to demonstrate the "practical" flexibility of our software. A first "proof" for these statements could be found by performing the DFG-benchmarks of the project "Flow Simulation with High Performance Computers".

Mainly active members of the FEAST-project are the 'numeric group' in Heidelberg around Rannacher and Turek. A list containing (almost) all participants (helping practically and theoretically) can be found in the Appendix.

FEATFLOW 1.1 is based on the FORTRAN77 finite element packages FEAT2D and FEAT3D which are not user oriented systems. They only provide subroutines for several main steps in a finite element program. The user should be familiar with the mathematical formulation of the discrete problems. The data structure of FEAT is transparent so that modifications or augmentations of the program package are very easy, for instance the implementation of new elements or the application of error control. For details concerning tools and data structures of these finite element codes the reader is referred to the manuals [3] and [9]. We consider to use other, more sophisticated finite element tools in FORTRAN90 under NETSCAPE in the future.

This manual has the following structure: Chapter 1 shows the mathematical background and explains the temporal and spatial discretizations. Additionally, subjects like nonlinear solution schemes, multigrid solvers for linear subproblems, discrete projection schemes for coupled systems, adaptive time step control and other optimization tools are treated explicitly. Inbetween, there is much more literature cited which explains all pointed subjects more in detail. We describe the discretization and solution process in form of flow charts since they can be directly translated into the programming structure of FEATFLOW.

This structure of programs and subroutines is explained in Chapter 2. We divide all software into three classes: Preprocessing, solver and postprocessing.

The preprocessing part contains the programs OMEGA2D (description of 2D domains, graphical generation of coarse tringulations, developed by Matthies/Schieweck in Magdeburg, and which will be soon replaced by our own JAVA–based preprocessing tool), TRIGEN2D (adaptively refined meshes in 2D, output of triangulations), TR2TO3 (generation of 3D meshes out of 2D meshes) and TRIGEN3D (analogous as TRIGEN2D but in 3D). These programs mainly provide (coarse) triangulations for the following solution and graphical output routines.

The solution part contains the solver packages PP2D/PP3D (as purely time dependent projection schemes) and CC2D/CC3D (solving stationary and nonstationary problems in a fully coupled way). The use of both solvers, two- as well as three–dimensional, is demonstrated and their input parameters are explained. The systems CP2D/CP3D are not finished yet and will be added to the next release (in fact, you may find in the recent version a test version of CP2D, together with some other 'new' software tools!).

The postprocessing part, finally, contains files and shell–scripts for supported graphical packages. These are, in this version, GNUPLOT (for 1D pictures) and MOVIE.BYU or CQUEL.BYU, resp., AVS for 2D- and 3D–graphics. Moreover, it has shown to be easy to incoorporate output files for GRAPE or some other appropriate graphics, and to add our own particle tracing tool PTRAC.

At the end of this chapter, we show the content of the other subdirectories of FEATFLOW and explain the structure of FEATFLOW with its installation, backup, makefile and library parts.

In addition, most important for the user, sample programs for some standard applications (installation, preprocessing and the solution of a 2D and 3D problem) are included in Chapter 3. These can be used as starting programs which may be modified for the actual application.

There are several main levels to be explained in detail:

- Installation of FEATFLOW, editing of makefiles and other preparation for the use
- Generation of coarse and refined triangulations for the solution process
- Input data (data files and input parameter, as for instance boundary conditions and right hand side)
- Code execution and explanation of output data
- Graphical presentation of the data

At the end of this manual, the Appendix contains a discussion of the "most usual" errors and problems with FEATFLOW and how to survive them, followed by a list with names and contact addresses of involved persons, and a remark how to get FEATFLOW. Additionally, there is a short section which shows projects being under development for future versions of FEATFLOW

We hope that you enjoy this software and are very grateful for every comment concerning bugs and critical aspects or subjects to be added.

In fact, this version FEATFLOW 1.1 contains only slight modifications compared to the 'original' version FEATFLOW 1.0! However, we may announce still for 1998 the new version FEATFLOW 2.0 which will contain the essentially improved solvers CP2D and some tools for Boussinesq and/or nonnewtonian flows, including also the improved pre- and processing tools together with (freely downloadable!!!) AVS EXPRESS modules for high–end visualization. For a mathematical description of these software releases we recommend the papers in our paper archive (see the following URL) or our book 'Efficient solvers for incompressible flow problems: An algorithmic approach in view of computational aspects' which will appear by Springer Verlag. Further, we hope to present a first version of our new FEAST package.

For more information about these project, take a look at the Internet–URL:

http://www.iwr.uni-heidelberg.de/~featflow

which contains most of our research activities, including the 'Virtual Album of Fluid Motion'.

Heidelberg, August 1998

1. Mathematical Background

1.1. Introduction

The following mathematical description for the implemented software is part of our paper archive (see the given URL) or of our book:

'*Efficient solvers for incompressible flow problems: An algorithmic approach in view of computational aspects*'

which will appear by Springer Verlag. There, the reader will find much more details.

We consider numerical solution techniques for the nonstationary (or stationary, without the term $\mathbf{u}_t$) incompressible Navier–Stokes equations,

$$\mathbf{u}_t - \nu\Delta\mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u} + \nabla p = \mathbf{f} \quad , \quad \nabla\cdot\mathbf{u} = 0 \quad , \quad \text{in } \Omega\times(0,T]\,, \tag{1.1}$$

for a given force $\mathbf{f}$ and viscosity ν, with any prescribed boundary values on the boundary $\partial\Omega$ (of Dirichlet- or Neumann–type, see [10]) and an initial condition at $t = 0$. Solving this problem numerically still seems to be a considerable task in the case of long time calculations and higher Reynolds numbers, particularly in 3D.

The corresponding discretized nonlinear systems of equations may be treated by a coupled approach in $\mathbf{u}$ and p (see [23]) which promises the best stability behaviour, but also entails the largest numerical effort. Another variant, known as projection method (see [7]), decouples pressure and velocity, which reduces the problem to the solution of a sequence of "simple" (scalar) problems. However, at the same time, it leads to smaller time steps due to the inherently more explicit character and often suffers from spurious pressure boundary layers.

These different approaches lead to a large variety of schemes all of which are occuring in practice since years (see [22] and [21]). Theoretical considerations could provide some ideas, concerning stability of these schemes, convergence rates for subproblems, necessary time step sizes, or qualitative behaviour for large Reynolds numbers, but a complete analysis or quantitative prediction is not possible even today. Therefore, the only way to make a judgement was to perform numerical tests, at least for some classes of problems which seem to be representative. This was done in [21].

What we reached is a (finite element) discretization and a solution procedure such that:

1) The finite element spaces for $\mathbf{u}$ *and* p *are stable, i.e., satisfy the LBB–condition ([8]).*

2) A robust and efficient coupled solver is available.

3) A robust and efficient solver of projection type is available.

4) An efficient nonlinear solution strategy is available.

5) An efficient time step control is available.

The method which seems to satisfy all these requirements consists of *discrete projection schemes* with nonconforming linear or *rotated multilinear* finite elements for $\mathbf{u}$ and piecewise constant approximations for p (see [15],[22],[21]). With this approach, we can develop very efficient solution schemes of both coupled and projection type, with a special nonlinear or linearized treatment of the advection. The resulting solutions are coincident (as soon as the time steps are small enough), and no spurious pressure oscillations occur. This approach is the basis of our following theoretical and numerical investigations.

1.2. Discretization and solution techniques in FEATFLOW

We first discretize the time derivative in the Navier–Stokes equations (1.1) by one of the usual time–stepping schemes, with prescribed boundary values for every time step.

Given $\mathbf{u}^n$ *and the time step* $k = t_{n+1} - t_n$*, then solve for* $\mathbf{u} = \mathbf{u}^{n+1}$ *and* $p = p^{n+1}$

$$\frac{\mathbf{u} - \mathbf{u}^n}{k} + \theta[-\nu\Delta\mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u}] + \nabla p = \mathbf{g}^{n+1} \quad , \quad \nabla\cdot\mathbf{u} = 0 \quad , \quad \text{in } \Omega\,, \tag{1.2}$$

with right hand side

$$\mathbf{g}^{n+1} := \theta\mathbf{f}^{n+1} + (1-\theta)\mathbf{f}^n - (1-\theta)[-\nu\Delta\mathbf{u}^n + \mathbf{u}^n\cdot\nabla\mathbf{u}^n]\,. \tag{1.3}$$

In the past, explicit time–stepping schemes have been commonly used in nonstationary flow calculations, but because of the severe stability problems inherent in this approach, the required small time steps prohibited the long time solution of really time–dependent flows. Due to the high stiffness, one seems to be limited to implicit schemes in the choice of time–stepping methods for solving this problem. Since implicit methods have become feasible, thanks to more efficient linear solvers, the schemes most frequently used are either the simple first–order Backward Euler–scheme (BE), with $\theta = 1$, or the second–order Crank–Nicolson–scheme (CN), with $\theta = 1/2$. These two methods belong to the group of *one–step–θ–schemes*. The CN–scheme occasionally suffers from unexpected instabilities because of its only weak damping property (not strongly A–stable), while the BE–scheme is of first order accuracy only. Another method which seems to have the potential to excel in this competition is the *Fractional–step–θ–scheme* (FS). It uses three different values for θ and for the time step k at each time level. For a realistic comparison we define a macro time step with $K = t_{n+1} - t_n$ as a sequence of 3 time steps of possibly (variable) size k. Then, in the case of the Backward Euler- or the Crank–Nicolson–scheme, we perform 3 substeps with the same θ as above and time step $k = K/3$.

For the Fractional–step–θ–scheme we proceed as follows. Choosing $\theta = 1-\frac{\sqrt{2}}{2}$, $\theta' = 1-2\theta$, and $\alpha = \frac{1-2\theta}{1-\theta}$, $\beta = 1-\alpha$, the macro time step $t_n \to t_{n+1} = t_n + K$ is split into three consecutive substeps (with $\tilde{\theta} := \alpha\theta K = \beta\theta' K$):

$$\begin{aligned}
[I + \tilde{\theta}N(\mathbf{u}^{n+\theta})]\mathbf{u}^{n+\theta} + \theta K\nabla p^{n+\theta} &= [I - \beta\theta K N(\mathbf{u}^n)]\mathbf{u}^n + \theta K\mathbf{f}^n \\
\nabla\cdot\mathbf{u}^{n+\theta} &= 0\,, \\
[I + \tilde{\theta}N(\mathbf{u}^{n+1-\theta})]\mathbf{u}^{n+1-\theta} + \theta' K\nabla p^{n+1-\theta} &= [I - \alpha\theta' K N(\mathbf{u}^{n+\theta})]\mathbf{u}^{n+\theta} + \theta' K\mathbf{f}^{n+1-\theta} \\
\nabla\cdot\mathbf{u}^{n+1-\theta} &= 0\,, \\
[I + \tilde{\theta}N(\mathbf{u}^{n+1})]\mathbf{u}^{n+1} + \theta K\nabla p^{n+1} &= [I - \beta\theta K N(\mathbf{u}^{n+1-\theta})]\mathbf{u}^{n+1-\theta} + \theta K\mathbf{f}^{n+1-\theta} \\
\nabla\cdot\mathbf{u}^{n+1} &= 0\,.
\end{aligned}$$

Here and in the following, we use the more compact form for the diffusive and advective part

$$N(\mathbf{u})\mathbf{u} := -\nu\Delta\mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u}\,. \tag{1.4}$$

Being a strongly A–stable scheme, the FS–method possesses the full smoothing property which is important in the case of rough initial or boundary values. Further, it contains only very little numerical dissipation which is crucial in the computation of non–enforced temporal oscillations in the flow. A rigorous theoretical analysis of the FS–scheme (see [11],[12]) applied to the Navier–Stokes problem establishes second order accuracy for this special choice of θ. Corresponding numerical tests are performed in [21]. Therefore, this scheme can combine the advantages of both the classical CN–scheme (2nd order accuracy) and the BE–scheme (strongly A–stable), but with the same numerical effort.

So, in each time step we have to solve nonlinear problems of the following type:

Given $\mathbf{u}^n$, *parameters* $k = k(t_{n+1})$, $\theta = \theta(t_{n+1})$ *and* $\theta_i = \theta_i(t_{n+1}), i = 1,\ldots,3$, *then solve for* $\mathbf{u} = \mathbf{u}^{n+1}$ *and* $p = p^{n+1}$

$$[I + \theta k N(\mathbf{u})]\mathbf{u} + k\nabla p = [I - \theta_1 k N(\mathbf{u}^n)]\mathbf{u}^n + \theta_2 k\mathbf{f}^{n+1} + \theta_3 k\mathbf{f}^n \quad , \quad \nabla\cdot\mathbf{u} = 0\,. \tag{1.5}$$

For spatial discretization, we choose a finite element approach. In setting up a finite element model of the Navier–Stokes equations, one starts with a variational formulation. On the finite mesh T_h (triangles, quadrilaterals or their analogues in 3D) covering the domain Ω with local element width h, one defines polynomial trial functions for velocity and pressure. These spaces $\mathbf{H}_h$ and L_h should lead to numerically stable approximations, as $h \to 0$, i.e., they should satisfy the *Babuska–Brezzi condition* with a mesh–independent constant γ (see [8]),

$$\min_{p_h\in L_h}\max_{\mathbf{v}_h\in\mathbf{H}_h}\frac{(p_h, \nabla\cdot\mathbf{v}_h)}{\|p_h\|_0\,\|\nabla\mathbf{v}_h\|_0} \geq \gamma > 0\,. \tag{1.6}$$

Many stable pairs of finite element spaces have been proposed in the literature. Our favorite candidate is a quadrilateral element which, in 2D, uses piecewise *rotated bilinear* shape functions for the velocities, spanned by $\langle x^2-y^2, x, y, 1\rangle$, resp., *rotated trilinear* shape functions in 3D, spanned by $\langle x^2-y^2, x^2-z^2, x, y, z, 1\rangle$, and piecewise constant pressure approximations (see Figure 1.1). The nodal values are the mean values of the

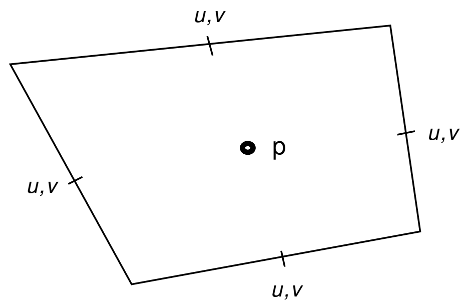

Figure 1.1: Nodal points of the nonconforming finite element pair in 2D

velocity vector over the element edges / faces (element type 30) or in the corresponding midpoints (element type 31), and the mean values of the pressure over the elements rendering this approach "nonconforming". This element is the natural quadrilateral analogue of the well–known triangular Stokes element of Crouzeix–Raviart (see [5]). A convergence analysis of both the parametric (elements `E030/E031`) and the nonparametric versions (elements `EM30/EM31`), which are preferrable on non–tensor product grids or meshes with large aspect ratios, is given in [15] and very promising computational results are reported in [19],[23],[21].

This element pair has several important features. It admits, beside the typical Galerkin–streamline diffusion technique, simple upwind strategies which lead to matrices with certain M–matrix properties. Further, efficient multigrid solvers are available which work satisfactorily over the whole range of relevant Reynolds numbers, $1 \leq Re \leq 10^5$, and also on nonuniform meshes. In [22], we have even shown by a complexity analysis that this pair of elements is most efficient compared to other finite element pairs, especially in the case of highly nonstationary flows. In combination with the *discrete projection methods*, see [22], it works very robust and efficient in a multigrid code also on highly stretched and anisotropic grids.

Using the same symbols $\mathbf{u}$ and p also for the coefficient vectors in the nodal representation for the functions $\mathbf{u}$ and p, the discrete version of problem (1.5) may be written as a (nonlinear) algebraic system of the form:

Given $\mathbf{u}^n$, a right hand side $\mathbf{g}$ and a time step k, then solve for $\mathbf{u} = \mathbf{u}^{n+1}$ and $p = p^{n+1}$

$$S\mathbf{u} + kBp = \mathbf{g} \quad , \quad B^T\mathbf{u} = 0\,, \tag{1.7}$$

with matrix S and right hand side $\mathbf{g}$ such that

$$S\mathbf{u} = [M + \theta k N(\mathbf{u})]\mathbf{u} \quad , \quad \mathbf{g} = [M - \theta_1 k N(\mathbf{u}^n)]\mathbf{u}^n + \theta_2 k \mathbf{f}^{n+1} + \theta_3 k \mathbf{f}^n\,. \tag{1.8}$$

Here, M is the *mass* matrix and $N(.)$ the *advection* matrix containing the diffusive and convective parts corresponding to the nonlinear form in (1.4). For dominant transport the advection part may include some stabilization, for instance, some upwind mechanism (see [23]) or the streamline diffusion method ([26]). B is the *gradient* matrix, and $-B^T$

the transposed *divergence* matrix. With M_l we denote the lumped mass matrix which is diagonal.

Two possible approaches for solving these discrete nonlinear problems are:

1) We first treat the nonlinearity by an outer nonlinear iteration of fixed point- or quasi–Newton type or by a linearization technique through extrapolation in time, and we obtain linear indefinite subproblems of Oseen type which can be solved by a coupled (versions CC2D/ CC3D*) or a splitting approach (versions* CP2D/ CP3D*).*

2) We first split the coupled problem and obtain definite problems in $\mathbf{u}$ *(Burgers–equations) as well as in* p *(linear pressure–Poisson-, resp., quasi–Poisson problems). Then we treat the nonlinear problems in* $\mathbf{u}$ *by an appropriate nonlinear iteration or a linearization technique (versions* PP2D/ PP3D*).*

In our applications the nonlinear problems are solved by the *adaptive fixed point defect correction method* (see [23]), while for the solution of the coupled problems, resp., for their decoupling, the *discrete projection method* formalism is used (see [22]). To understand completely the algorithm, we will shortly present other important tools of the solver, namely the multilevel solvers for the corresponding linear subproblems (Oseen problems, (convection–diffusion problems for $\mathbf{u}$, quasi–Poisson problems for p), the adaptive time step control and defect optimization procedures. In compact form the components of the solvers are explained in the following subsections.

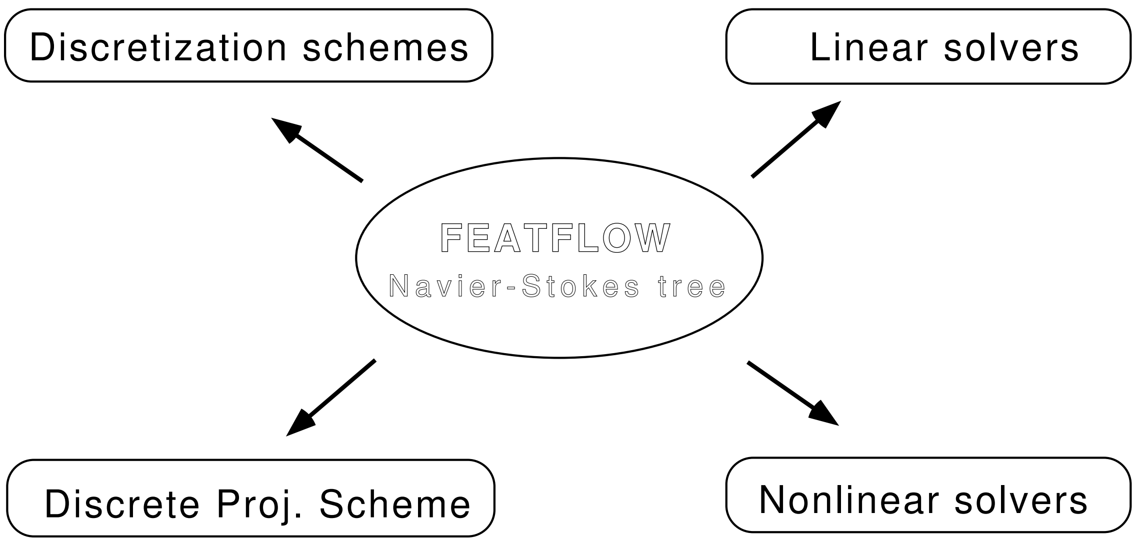

Figure 1.2: The solver structure of FEATFLOW

1.2.1. Discretization schemes

Spatial discretization

- nonconforming *nonparametric rotated bi/trilinear* finite elements with edge oriented d.o.f.'s for velocity
- piecewise constant finite elements for pressure
- a–priori adapted coarse meshes with exact boundary adaptation during successive refinements
- adaptive refinement leading to conforming triangulations possible
- adaptive FE-upwinding for convective terms (Samarskij upwinding)
- adaptive streamline–diffusion for convective terms
- natural *do nothing* b.c.'s
- *flux*- and *pressure drop* b.c.'s

Temporal discretization

- *One–step–θ–scheme* (Implicit Euler, Crank–Nicolson scheme)
- *Fractional–step–θ–scheme* as strongly A–stable implicit time stepping scheme of 2nd order
- adaptive time stepping by estimating the local discretization error (3 substeps with Δt and 1 substep with $3\Delta t$)
- extrapolation of partial solutions for higher accuracy

1.2.2. Nonlinear solvers

Nonlinear iteration

- for stationary and nonstationary problems of Navier–Stokes- or Burgers type
- fixed point iteration (quasi–Newton) with *adaptive step length control* by nonlinear defect minimization
- additional defect correction possible

Linearization

- for nonstationary problems only
- semi–implicit treatment of nonlinear convective terms by linear extrapolation (in time of 2nd order)

1.2.3. Linear solvers

Multigrid for velocity and pressure simultanously (Oseen):

- *nonconforming mesh adapted makro-elementwise* interpolation for grid–transfer
- *adaptive step length control* for correction step (with F–cycle)
- *Vanca*–like block–Gauß-Seidel scheme as smoother and solver

Multigrid for velocity (Burgers) and pressure (quasi–Poisson):

- *nonconforming mesh adapted makro-elementwise* interpolation for grid–transfer
- *adaptive step length control* for correction step (with F–cycle)
- ILU/SOR–scheme as smoother with special renumbering strategies

1.2.4. Discrete projection method

For linear (or nonlinear) coupled problems:

$$S\mathbf{u} + kBp = \mathbf{g} \quad , \quad B^T\mathbf{u} = 0$$

Given p^0 , solve for $l = 1, \ldots, L$:

$$p^l = p^{l-1} - \alpha^l [B^T C^{-1} B]^{-1} (B^T S^{-1} B p^{l-1} - \frac{1}{k} B^T S^{-1} \mathbf{g})$$

Set $p := p^L$, and determine u through $(\Rightarrow B^T\mathbf{u} = 0)$

$$S\mathbf{u} = \mathbf{g} - kBp + \frac{k}{\alpha^l}[\alpha^l I - SC^{-1}]B(p^L - p^{L-1})$$

Classical Richardson–scheme for Schur–complement formulation with preconditioner $P^{-1} = [B^T C^{-1} B]^{-1}$

$C = S \quad , \quad L = 1$ for CC2D/ CC3D

$C = M_l \quad , \quad L \geq 1$ for PP2D/ PP3D

If $C = M_l \;\Rightarrow\; P$ corresponds to discrete Laplacian with minimal matrix stencil (5 in 2D/7 in 3D) independent of the mesh

Further improvements:

- Additional preconditioner possible (of diffusive type)
- Elimination of pressure boundary layers
- *Multilevel Discrete Projection scheme* (CP2D/ CP3D)

$\longrightarrow$ "fastest" solver/discretization for fully nonstationary problems
$\longrightarrow$ "robust" solver/discretization for quasi–stationary problems

All proposed discretization and solution steps can be represented by the following flow chart ("the Navier-Stokes tree"), leading to the methods mentioned above. It is remarkable that most of all existing Navier-Stokes solvers can be interpreted by this tree structure. This is a very powerfull tool for the understanding and, then, optimization of existing algorithms and software.

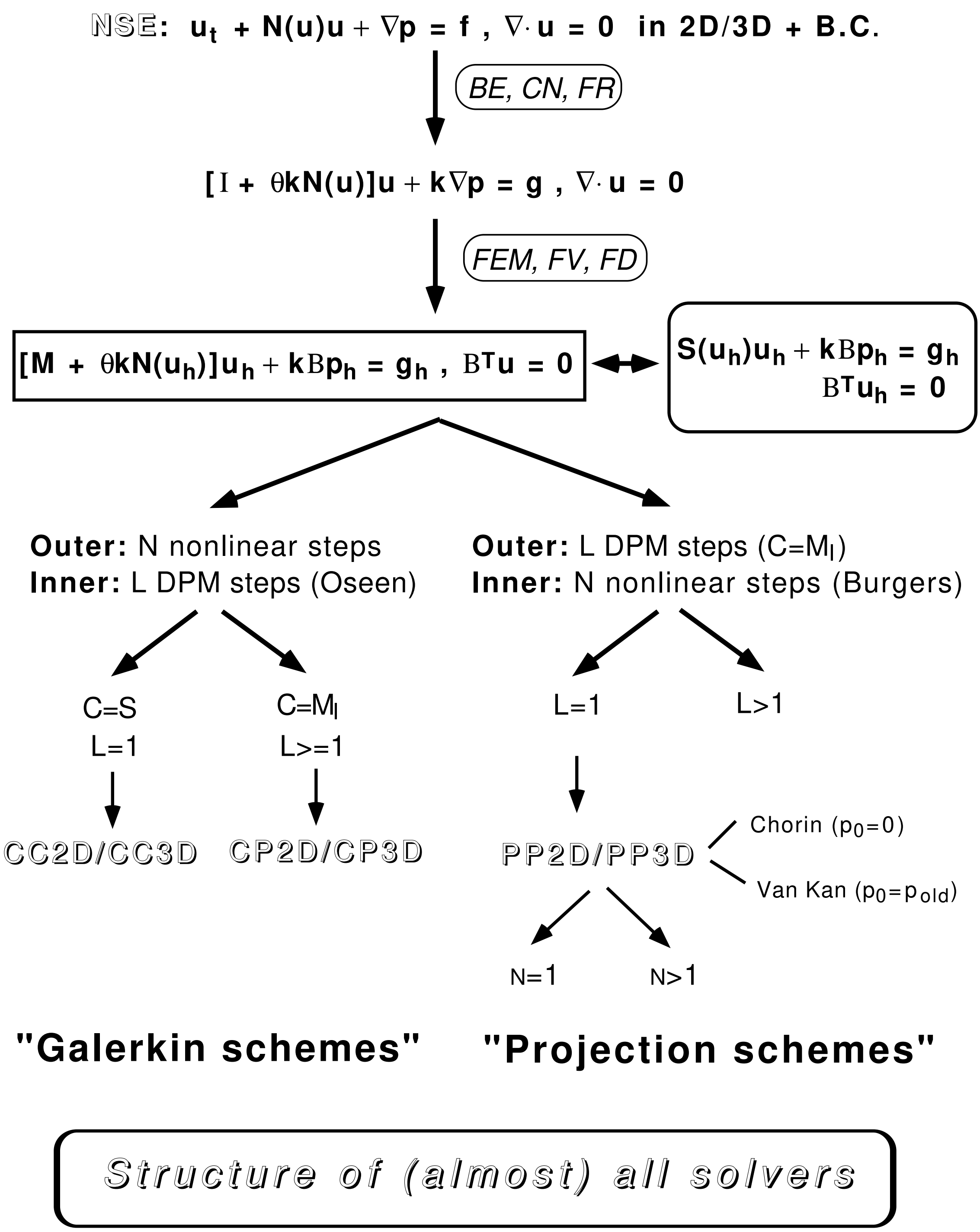

Figure 1.3: The Navier-Stokes tree

1.3. The Navier–Stokes solver packages in FEATFLOW

The different solution schemes for the Navier–Stokes equations we can apply are the following ones:

Coupled solver CC2D/ CC3D:

We use the *adaptive fixed point defect correction method* as outer iteration, while for the linear coupled subproblems we choose $C = S$ and $\alpha = 1$. Hence, the linear equations in the nonlinear process are solved in "one iteration step", meaning that $L = 1$. This solver is a multilevel based approach (see [19]) with a block Gauß-Seidel–scheme as smoothing operation (Vanca–smoother). There may be problems on very anisotropic meshes, and there is no gain in efficiency for fully nonstationary problems if $\Delta t \to 0$. This solver is, actually, preferrable for the case of low Reynolds numbers (stationary or quasi–stationary problems).

Mixed solver CP2D/ CP3D:

We use again the *adaptive fixed point defect correction method* as outer iteration and select the preconditioner $C = M_l$. We will obtain the same solutions as by CC2D/ CC3D, however in a more robust way. First tests show that this solver works in a very efficient way, independent of the mesh and Re number, being very efficient for fully nonstationary problems. The essential tool is a so called "multilevel discrete projection" algorithm for the linear coupled subproblems. This solver will be added to FEATFLOW in the next release.

Projection solver PP2D/ PP3D:

First, we apply a decoupling step for $\mathbf{u}$ and p as outer iteration ("nonlinear discrete projection scheme"). We choose again $C = M_l$, but perform only $L = 1$ iteration in each time step. In the fully nonlinear case, we use the same fixed point iteration, as explained above, for the transport–diffusion–step with nonlinear operator S ("Burgers equations"). In an analogous way, we treat (by extrapolation) the linearized schemes.

All versions of CC2D/ CC3D and CP2D/ CP3D (and even PP2D/ PP3D for L large enough) lead to the "same" solutions if the numbers of nonlinear steps N and discrete projection steps L are large enough. The complexity analysis in [22] shows that in 2D one single iteration of the coupled scheme ($C = S$) can be expected to cost at least 10 (2D) until 20 (3D) times more than one iteration of the operator splitting method ($C = M_l$). At the same time, we have to use more nonlinear sweeps or smaller time steps for $C = M_l$, due to the more explicit character of the scheme. However, extensive numerical tests in [21] show that the projection solvers are superior in most cases compared to CC2D/ CC3D, especially for highly nonstationary flows.

In the next chapter one will see that FEATFLOW follows exactly this flow chart ("the Navier-Stokes tree") and that the programming structure is a direct translation of this tree structure.

2. Structure of FEATFLOW

2.1. The programming structure of FEATFLOW

As pointed in the last section, the programming structure of FEATFLOW follows directly the tree structure in diagram 1.3.

The corresponding programs CC2D/ CC3D and PP2D/ PP3D (CP2D/ CP3D are not yet finished) are the main programs which form the body of the code. Their task is to initialize all data, to build triangulations, linear matrices, pointer structures and right hand sides on all grid levels (all in the file `init1.f`) and to generate all vectors needed (also in `init1.f`). After selecting all parameters for the chosen time stepping scheme and corresponding adaptive time step size, the discretization process is finished and the time stepping loop may begin. That means, everything is prepared for solving the pointed nonlinear coupled generalized stationary Navier–Stokes problems corresponding to each time level. After doing this in the subroutine `prostp.f`, resp., `mgstp.f`, the main program `cc2d.f` (or `cc3d.f`, `pp2d.f`, `pp3d.f`) continues with the time step control, selecting a new time step size for the next (macro) step or repeating the last one if necessary, then evaluates the actual solution vector with some extrapolation in time if needed, and enters the output–subroutines `fpost.f` or `error.f`. These main programs provide most of the output for monitoring all actions and give the needed workspace and computer times for the performed application.

The subroutine on the next level is `prostp.f`, resp., `mgstp.f`, solving (or approximating only in the case of the pure projection schemes PP2D/ PP3D) a number of generalized stationary Navier–Stokes equations corresponding to the actual macro time level. The number of these problems and their parameters depend on the scheme and the time step control chosen. Furthermore, the corresponding boundary conditions are set in this file (by `bndry.f`).

As shown in the previous tree diagram, at this point there are the differences between the codes PP2D/ PP3D and CC2D/ CC3D. While the fully coupled versions CC2D/ CC3D perform an outer linearization by applying first the adaptive fixed point control as non-linear solver, PP2D/ PP3D first decouple velocity and pressure, and then solve a nonlinear Burgers–equation <u>and</u> a linear quasi–Poisson problem.

In detail, PP2D/ PP3D do the following for "solving" one nonlinear coupled equation: With given pressure as right hand side, a nonlinear Burgers–equation is solved in `nsdef.f`, with adaptive defect optimization (`optcnl.f`) and multigrid solvers for the linear convection–-diffusion problems (`m011.f`). With the divergence of this intermediate velocity as right hand side a corresponding update problem of quasi–Poisson type for the pressure is solved, by a multigrid scheme (again `m011.f`), and the auxiliary solution is added to the old pressure to obtain the new one. Finally, the diffusive preconditioner may be additionally obtained to improve the new pressure. This approach corresponds to the special version $L = 1$ of the nonlinear discrete projection scheme. We perform this decoupling process only once, and, hence, we obtain an approximate solution in each time step only. However, that is very similar to the idea of the (classical) projection schemes (of Van Kan [27], resp., Chorin [4]).

In contrast, CC2D/ CC3D (the upcoming CP2D/ CP3D analogously) perform one nonlinear coupled solution step in a different way: The file `nsdef.f` handles the solution process being a solver of the nonlinear coupled equation, with performing Oseen equations (linearized Navier–Stokes equations) as preconditioner in each nonlinear step (`m011.f`, with a special block Gauss-Seidel smoother), and following defect minimization (`optcnl.f`). The versions CP2D/ CP3D will differ in solving the linearized coupled equations by a special multilevel scheme involving the linear discrete projection method ("multilevel discrete projection method").
This approach guarantees the fully coupled solution and, hence, a larger stability and improved accuracy compared to PP2D/ PP3D. However, the numerical effort increases and, in this actual release, CC2D/ CC3D is preferrable in the case of low Reynolds numbers and on moderate meshes only.

Following these remarks the programming structure of CC2D/ CC3D and PP2D/ PP3D can be represented by the following diagrams.

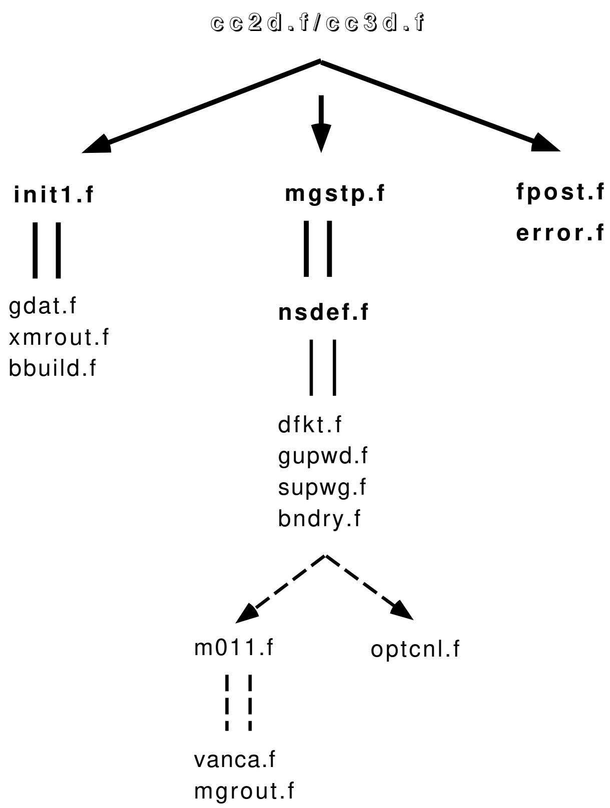

Figure 2.1: The CC2D/ CC3D structure

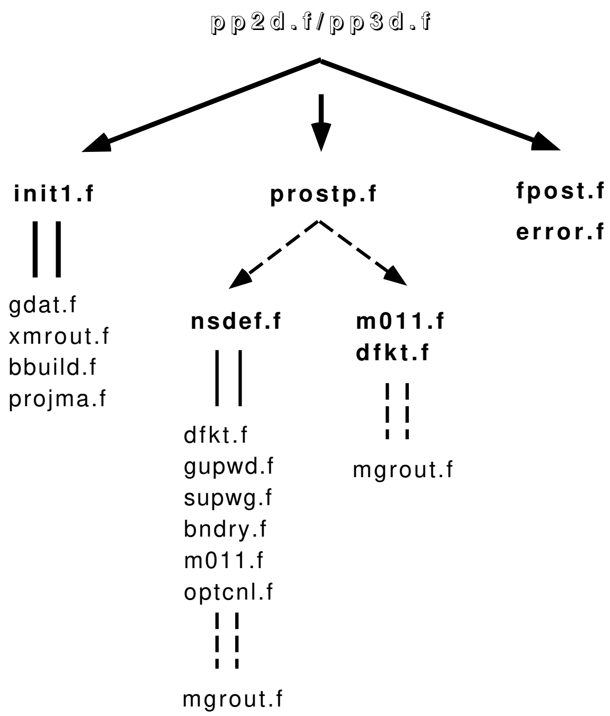

Figure 2.2: The PP2D/ PP3D structure

Beside the explained solver tools there are some other programs belonging to FEATFLOW which are essential for the preprocessing (description of domain Ω, coarsest triangulation, pre–adapted meshes, output for graphics) and for the preparation of an application (generation and interpolation of a start solution). These are in detail:

<u>OMEGA2D:</u>

OMEGA2D is a graphical preprocessor which provides tools for describing and defining the boundary components of a 2D domain Ω and for generating a first (and very coarse) triangulation of this domain. This process is fully interactive and mouse–oriented, however, actually, running under DOS and WINDOWS 3.1 only. It has been created by Matthies/ Schieweck in Magdeburg, and further development is under progress. In the next chapter we demonstrate how to use it, and the complete manual of Version 1.6 can be found in the `manual` directory of FEATFLOW.

<u>TRIGEN2D:</u>

TRIGEN2D is a preprocessing tool for designing "better" 2D coarse triangulations and to write the corresponding data in some special formats onto hard disc. It reads in a parametrization (in standard FEAT- or in OMEGA2D–format) and a corresponding mesh (in FEAT–format), and refines it locally in a successive manner. The user has different possibilities for mesh refinement, and in this release the elements to be refined have to be given in a list. We hope to provide a graphical interface for this marking process in the next version, and, additionally, to use this tool in a fully adaptive mesh refinement process. Furthermore, a boundary check on consistency for complex domains is included. The formats for data–output are: formatted or unformatted FEAT–style, BYU–style and AVS-ucd–format.

TR2TO3:

TR2TO3 is a tool providing 3D mesh generation from 2D meshes, all in FEAT–format. This tool is designed for applications when the 3D mesh is simply constructed by popping 2D mesh layers in a sandwich–like technique. For instance, problems in ducts around cylinders or other "prism"-like bodies are belonging to this class (see the example in the next chapter).

TRIGEN3D:

TRIGEN3D is doing the same job as TRIGEN2D, but in 3 dimensions. However, up to now, only FEAT–format is supported (that means, the boundary of Ω must be defined by the standard FEAT parametrization file). We hope to add some appropriate CAD–tools in future, for being able to model much more complex domains. Additionally, the adaptive grid refinement procedure is not finished yet, and will be part of the next version. However, at least the data–output is performing analogously as in 2D.

INTPOL2D and INTPOL3D:

These programs will be tools for interpolating a given solution vector on a given mesh to an arbitrary different triangulation, in 2D as well as in 3D. Both data and grid have to be given in FEAT–format. These tools will be available in the next release.

2.2. The input data of FEATFLOW

In this section, we explain all input data which are needed to start an application with FEATFLOW. Let's start with the preprocessing tools, and with some general comments. In the next chapter, we demonstrate explicitely the meaning of all input data.

2.2.1. General comments for user–input

First of all, files containing pure input parameter for user applications have to be in the directory `#data`. The file name is the name of the corresponding program, ended with `.dat`, for instance `#data/pp2d.dat`. These files will be explained in detail in the following subsections. Examples for different "styles" of parameter parameter files, concerning robustness, efficiency and small stoarge amount, can be found in the directory `application /data_example`.

Furthermore, the only source–files which have to be modified (by editing or copying) are the parametrization files `parq2d.f`, resp., `parq3d.f`, and the data files `indat2d.f`, resp., indat3d.f. They contain all information about boundary values, right hand sides, exact solutions and their derivatives and output constants like lift and drag or integral or pointwise values in some special coordinates. The last file which has to be edited is a corresponding .inc–file, for instance pp2d.inc, containing the needed storage size for the applied program (for instance, NNWORK=1.000.000 means 1 million double precision elements needed, or, resp., a storage amount of 8 Mbyte). These files are usually located in the directory input_files. The best way is to collect all data files together in a subdirectory, corresponding to the actual application. This is usually done by FEATFLOW, for instance in the directories application/user_start, application/example and application/comp. Furthermore, the corresponding makefiles are also located in input_files, such that all data files are concentrated at one place.

2.2.2. indat2d.f/indat3d.f/parq2d.f/parq3d.f – data files for user

<u>indat2d.f:</u>

```
************************************************************************
      DOUBLE PRECISION FUNCTION FDATIN(ITYP,IBLOC,X,Y,TIMENS,RE)
*
*     Prescribed data for files coeff.f and bndry.f
************************************************************************
      IMPLICIT DOUBLE PRECISION(A,C-H,O-U,W-Z),LOGICAL(B)
      PARAMETER (PI=3.1415926535897931D0)
C
      FDATIN=0D0
C
C=======================================================================
C *** Case 1: Velocity boundary values and/or exact solution
C=======================================================================
C
      IF (ITYP.EQ.1) THEN
C
       IF (IBLOC.EQ.1) THEN
        IF (X.EQ.0D0) FDATIN=4D0*0.3D0/0.1681D0*Y*(0.41D0-Y)
     *                          *SIN(0.5D0*PI*MIN(TIMENS,1D0)/1D0)
       ENDIF
C
       IF (IBLOC.EQ.2) THEN
        IF (X.EQ.0.0D0) FDATIN=0D0
       ENDIF
C
      ENDIF
C
C=======================================================================
C *** Case 2: Velocity x-derivative of exact solution
C=======================================================================
C
      IF (ITYP.EQ.2) THEN
C
       IF (IBLOC.EQ.1) THEN
        IF (X.EQ.0.0D0) FDATIN=0D0
       ENDIF
C
       IF (IBLOC.EQ.2) THEN
        IF (X.EQ.0.0D0) FDATIN=0D0
       ENDIF
C
      ENDIF
C
C=======================================================================
C *** Case 3: Velocity y-derivative of exact solution
```

```
C=======================================================================
C
      IF (ITYP.EQ.3) THEN
C
       IF (IBLOC.EQ.1) THEN
        FDATIN=0D0
       ENDIF
C
       IF (IBLOC.EQ.2) THEN
        FDATIN=0D0
       ENDIF
C
      ENDIF
C
C=======================================================================
C *** Case 4: Exact pressure solution
C=======================================================================
C
      IF (ITYP.EQ.4) THEN
C
       FDATIN=0D0
C
      ENDIF
C
C=======================================================================
C *** Case 5: Right hand side for momentum equation
C=======================================================================
C
      IF (ITYP.EQ.5) THEN
C
       IF (IBLOC.EQ.1) THEN
        FDATIN=0D0
       ENDIF
C
       IF (IBLOC.EQ.2) THEN
        FDATIN=0D0
       ENDIF
C
      ENDIF
C
C=======================================================================
C *** Case 6: Right hand side for continuity equation
C=======================================================================
C
      IF (ITYP.EQ.6) THEN
C
       FDATIN=0D0
C
      ENDIF
C
C=======================================================================
C *** Case 7: Mean pressure values
C=======================================================================
C
      IF (ITYP.EQ.7) THEN
       DPAR=X
       INPR=IBLOC
C
       IF ((DPAR.GT.1D0).AND.(DPAR.LT.2D0).AND.(INPR.EQ.1)) THEN
        FDATIN=0D0
       ENDIF
C
       IF ((DPAR.GT.3D0).AND.(DPAR.LT.4D0).AND.(INPR.EQ.1)) THEN
        FDATIN=0D0
       ENDIF
C
      ENDIF
C
99999 END
C
```

```
************************************************************************
      SUBROUTINE NEUDAT(INPART,INPRN,DPARN1,DPARN2,TIMENS)
*
*     Neumann-boundary part
************************************************************************
      IMPLICIT DOUBLE PRECISION(A,C-H,O-U,W-Z),LOGICAL(B)
C
C=======================================================================
C *** Case 0: Set number of Neumann-boundary parts
C=======================================================================
C
      IF (INPART.EQ.0) THEN
       INPART=1
      ENDIF
C
C=======================================================================
C *** Case <>0: Specify Neumann-boundary parts
C=======================================================================
C
      IF (INPART.GT.0) THEN
C
       IF (INPART.EQ.1) THEN
        DPARN1=1D0
        DPARN2=2D0
        INPRN =1
       ENDIF
C
      ENDIF
C
99999 END
C
************************************************************************
      SUBROUTINE PTSDAT(TIMENS,DNU)
*
*     Data for Point-output (for fpost and bdpres and bdforc)
************************************************************************
      IMPLICIT DOUBLE PRECISION(A,C-H,O-U,W-Z),LOGICAL(B)
      COMMON /NSPTS/  KPU(2),KPP(4),KPX(4),KPI(2),DPI(2,2),DPF(2)
      SAVE
C
      EXTERNAL UE
C
C=======================================================================
C *** Points for velocity, pressure and flux-tracing
C=======================================================================
C
      KPU(1)=22
      KPU(2)=2
C
      KPP(1)=42
      KPP(2)=46
      KPP(3)=2
      KPP(4)=22
C
      KPX(1)=41
      KPX(2)=10
C
C=======================================================================
C *** Parameters for 2 integral pressures in bdpres.f
C=======================================================================
C
      KPI(1)  =2
      DPI(1,1)=0D0
      DPI(2,1)=1D0
C
      KPI(2)  =2
      DPI(1,2)=0.00D0
      DPI(2,2)=0.25D0
C
C=======================================================================
```

```
C *** Parameters for lift (DFW) and drag (DAW) in bdforc.f (INPR=2)
C ***
C *** dfw=2 int_s [dpf(1) dut/dn n_y - p n_x] ds / dpf(2)
C *** daw=2 int_s [dpf(1) dut/dn n_x + p n_y] ds / dpf(2)
C ***
C=====================================================================
C
      RHO  =1.0D0
      DIST =0.1D0
      UMEAN=0.2D0
C
      DPF(1)=RHO*DNU
      DPF(2)=RHO*DIST*UMEAN**2
C
99999 END
```

Most command lines have not to be explained more in detail (even being written in FORTRAN77 they explain themselves); only a few comments are necessary:

In the case `IF (ITYP.EQ.7) THEN ...` in `FDATIN` the mean pressure values on boundary parts corresponding to natural b.c.'s can be described. This is done by prescribing the starting parameter value (`DPAR1`) and the ending parameter value (`DPAR2`) corresponding to the parametrization which is used for describing the geometrical boundary. Additionally, the number of the boundary component has to be defined (`INPR`).

In `NEUDAT` the boundary parts containing natural boundary conditions are defined. This is done by prescribing the number of such boundary parts (by setting `INPART`), and then again by setting the starting parameter value (`DPARN1`), the ending parameter value (`DPARN2`) and the corresponding boundary component (`INPRN`).

In `PTSDAT` certain mesh points are defined in which some velocity components (`KPU(1)`, `KPU(2)`), some pressure values (`KPP(1),KPP(2),KPP(3),KPP(4)`) and a flux value (defined as difference of streamfunction values `KPX(1)` and `KPX(2)`) are printed for the runtime protocol. Furthermore, 2 integral pressure values on some fixed boundary parts (parameter values `DPI(1,1),DPI(2,1)` and boundary component `KPI(1)`, resp., `DPI(1,2),DPI(2,2)` and `KPI(2)`) are defined appropriately. And finally, some constants for the calculation of lift and drag on boundary component 2 (!) are prescribed. If not desired, set `DPF(1)`, `DPF(2)` to 0, or `KPI(1)`, `KPI(2)` to 0.

indat3d.f:

```
************************************************************************
      DOUBLE PRECISION FUNCTION FDATIN(ITYP,IBLOC,X,Y,Z,TIMENS,RE)
*
*     Prescribed data for files coeff.f and bndry.f
************************************************************************
      IMPLICIT DOUBLE PRECISION(A,C-H,O-U,W-Z),LOGICAL(B)
      PARAMETER (PI=3.1415926535897931D0)
C
      FDATIN=0D0
C
C=======================================================================
C *** Case 1: Velocity boundary values and/or exact solution
C=======================================================================
C
      IF (ITYP.EQ.1) THEN
C
       IF (IBLOC.EQ.1) THEN
        IF (X.EQ.0D0)
     *      FDATIN=16D0*0.45D0/(0.41D0**4)*Y*(0.41D0-Y)*Z*(0.41D0-Z)
       ENDIF
C
       IF (IBLOC.EQ.2) THEN
        IF (X.EQ.0.0D0) FDATIN=0D0
       ENDIF
C
       IF (IBLOC.EQ.3) THEN
        IF (X.EQ.0.0D0) FDATIN=0D0
       ENDIF
C
      ENDIF
C
C=======================================================================
C *** Case 2: Velocity x-derivative of exact solution
C=======================================================================
C
      IF (ITYP.EQ.2) THEN
C
       IF (IBLOC.EQ.1) THEN
        IF (X.EQ.0.0D0) FDATIN=0D0
       ENDIF
C
       IF (IBLOC.EQ.2) THEN
        IF (X.EQ.0.0D0) FDATIN=0D0
       ENDIF
C
       IF (IBLOC.EQ.3) THEN
        IF (X.EQ.0.0D0) FDATIN=0D0
       ENDIF
      ENDIF
C
C=======================================================================
C *** Case 3: Velocity y-derivative of exact solution
C=======================================================================
C
      IF (ITYP.EQ.3) THEN
C
       IF (IBLOC.EQ.1) THEN
        FDATIN=0D0
       ENDIF
C
       IF (IBLOC.EQ.2) THEN
        FDATIN=0D0
       ENDIF
C
       IF (IBLOC.EQ.3) THEN
        FDATIN=0D0
       ENDIF
```

```
C
      ENDIF
C
C=======================================================================
C *** Case 4: Velocity z-derivative of exact solution
C=======================================================================
C
      IF (ITYP.EQ.4) THEN
C
       IF (IBLOC.EQ.1) THEN
        FDATIN=0D0
       ENDIF
C
       IF (IBLOC.EQ.2) THEN
        FDATIN=0D0
       ENDIF
C
       IF (IBLOC.EQ.3) THEN
        FDATIN=0D0
       ENDIF
C
      ENDIF
C
C=======================================================================
C *** Case 5: Exact pressure solution
C=======================================================================
C
      IF (ITYP.EQ.5) THEN
C
       FDATIN=0D0
C
      ENDIF
C
C=======================================================================
C *** Case 6: Right hand side for momentum equation
C=======================================================================
C
      IF (ITYP.EQ.6) THEN
C
       IF (IBLOC.EQ.1) THEN
        FDATIN=0D0
       ENDIF
C
       IF (IBLOC.EQ.2) THEN
        FDATIN=0D0
       ENDIF
C
       IF (IBLOC.EQ.3) THEN
        FDATIN=0D0
       ENDIF
C
      ENDIF
C
C=======================================================================
C *** Case 7: Right hand side for continuity equation
C=======================================================================
C
      IF (ITYP.EQ.7) THEN
C
       FDATIN=0D0
C
      ENDIF
C
C=======================================================================
C *** Case 8: Mean pressure values
C=======================================================================
C
      IF (ITYP.EQ.8) THEN
C
       FDATIN=0D0
```

```
C
      ENDIF
C
99999 END
C
************************************************************************
      SUBROUTINE NEUDAT(IEL,INPR,PX,PY,PZ,TIMENS,IFLAG)
*
*     Neumann-boundary part
************************************************************************
      IMPLICIT DOUBLE PRECISION(A,C-H,O-U,W-Z),LOGICAL(B)
C
C=======================================================================
C *** Set Neumann-boundary parts
C=======================================================================
C
      IF (PX.EQ.2.5D0) THEN
       IFLAG=1
      ENDIF
C
99999 END
C
************************************************************************
      SUBROUTINE BDPDAT(IEL,INPR,PX,PY,PZ,TIMENS,IFLAG1,IFLAG2)
*
*     Pressure integral boundary part
************************************************************************
      IMPLICIT DOUBLE PRECISION(A,C-H,O-U,W-Z),LOGICAL(B)
C
C=======================================================================
C *** Set pressure integral boundary parts
C=======================================================================
C
      IF (((PZ.GT.0.00D0).AND.(PZ.LT.0.41D0)).AND.
     *    ((PX.GE.0.45D0).AND.(PX.LE.0.55D0)).AND.
     *    ((PY.GE.0.15D0).AND.(PY.LE.0.25D0))) THEN
       IFLAG1=1
      ENDIF
C
      IF (((PZ.GT.0.00D0).AND.(PZ.LT.0.41D0)).AND.
     *     (PX.EQ.0.45D0)                   .AND.
     *    ((PY.GE.0.15D0).AND.(PY.LE.0.25D0))) THEN
       IFLAG2=1
      ENDIF
C
99999 END
C
************************************************************************
      SUBROUTINE BDFDAT(IEL,INPR,PX,PY,PZ,TIMENS,DNU,IFLAG,DPF1,DPF2)
*
*     lift and drag data
************************************************************************
      IMPLICIT DOUBLE PRECISION(A,C-H,O-U,W-Z),LOGICAL(B)
C
C=======================================================================
C *** Parameters for lift (DFW) and drag (DAW) in bdforc.f (INPR=2)
C ***
C *** dfw=2 int_s [dpf(1) dut/dn n_y - p n_x] ds / dpf(2)
C *** daw=2 int_s [dpf(1) dut/dn n_x + p n_y] ds / dpf(2)
C ***
C=======================================================================
C
      RHO  =1.0D0
      DIST =0.041D0
      UMEAN=0.2D0
C
      DPF1=RHO*DNU
      DPF2=RHO*DIST*UMEAN**2
C
C=======================================================================
```

```
C
      IF (((PZ.GT.0.00D0).AND.(PZ.LT.0.41D0)).AND.
     *    ((PX.GE.0.45D0).AND.(PX.LE.0.55D0)).AND.
     *    ((PY.GE.0.15D0).AND.(PY.LE.0.25D0))) THEN
       IFLAG=1
      ENDIF
C
99999 END
C
************************************************************************
      SUBROUTINE PTSDAT(TIMENS,DNU)
*
*     Data for Point-output (for fpost)
************************************************************************
      IMPLICIT DOUBLE PRECISION(A,C-H,O-U,W-Z),LOGICAL(B)
      COMMON /NSPTS/  KPU(2),KPP(4)
      SAVE
C
C=======================================================================
C *** Points for velocity and pressure
C=======================================================================
C
      KPU(1)=7263
      KPU(2)=3444
C
      KPP(1)=3491
      KPP(2)=3557
      KPP(3)=3575
      KPP(4)=3444
C
99999 END
```

In `NEUDAT` the boundary parts containing natural boundary conditions are defined. This is done by prescribing the cartesian coordinates for this manifold. Analogously, in `BDPDAT` the coordinates for the pressure integral boundary parts are set, and in `BDFDAT` the coordinates for calculating lift and drag. If not desired, set the corresponding `IFLAG` parameters to 0.

Again, in `PTSDAT` certain mesh points are defined in which some velocity components (`KPU(1), KPU(2)`) and some pressure values (`KPP(1),KPP(2),KPP(3),KPP(4)`) are printed for the runtime protocol.

parq2d.f:

This file is either a standard FEAT parametrization file which is created by your own (see the FEAT2D manual) or the special OMEGA2D parametrization file. In this case, the file `parpre.f` has to be copied onto `parq2d.f`.

parq3d.f:

```
************************************************************************
      DOUBLE PRECISION FUNCTION PARX(T1,T2,T3,IBCT)
      IMPLICIT REAL*8 (A-H,O-Z)
      PARX=T1
99999 END
C
      DOUBLE PRECISION FUNCTION PARY(T1,T2,T3,IBCT)
      IMPLICIT REAL*8 (A-H,O-Z)
      PARY=T2
99999 END
C
      DOUBLE PRECISION FUNCTION PARZ(T1,T2,T3,IBCT)
      IMPLICIT REAL*8 (A-H,O-Z)
      PARZ=T3
      GOTO 99999
99999 END
C
************************************************************************
      SUBROUTINE   TRPARV (DCORVG,KNPR,KVEL,NVT,NVEL)
      IMPLICIT DOUBLE PRECISION (A,C-H,O-U,W-Z),LOGICAL(B)
      DIMENSION DCORVG(3,*),KNPR(*),KVEL(NVEL,*)
C
C-----------------------------------------------------------------------
C
      DO 10 IVT=1,NVT
      INPR=KNPR(IVT)
      IF (INPR.EQ.0) GOTO 10
C
      IEL=KVEL(4,IVT)
      PX=DCORVG(1,IVT)
      PY=DCORVG(2,IVT)
      PZ=DCORVG(3,IVT)
C
      IF ((ABS(PZ-0.0D0).GT.1D-8).AND.(ABS(PZ-0.41D0).GT.1D-8).AND.
     *    (ABS(PX-0.0D0).GT.1D-8).AND.(ABS(PX-2.50D0).GT.1D-8).AND.
     *    (ABS(PY-0.0D0).GT.1D-8).AND.(ABS(PY-0.41D0).GT.1D-8)) THEN
       PXM=0.50D0
       PYM=0.20D0
       RAD=0.05D0
       DL=SQRT((PX-PXM)**2+(PY-PYM)**2)
       DCORVG(1,IVT)=PXM+RAD/DL*(PX-PXM)
       DCORVG(2,IVT)=PYM+RAD/DL*(PY-PYM)
       GOTO 10
      ENDIF
C
C
      IF ((ABS(PZ-0.0D0).LE.1D-8).OR.(ABS(PZ-0.41D0).LE.1D-8)) THEN
       PXM=0.50D0
       PYM=0.20D0
       RAD=0.05D0
       RADH=0.05D0+1D-8
       IF ((ABS(PX-PXM).LE.RADH).AND.(ABS(PY-PYM).LE.RADH).AND.
     *     (IEL.EQ.0)) THEN
        DL=SQRT((PX-PXM)**2+(PY-PYM)**2)
        DCORVG(1,IVT)=PXM+RAD/DL*(PX-PXM)
        DCORVG(2,IVT)=PYM+RAD/DL*(PY-PYM)
        GOTO 10
       ENDIF
      ENDIF
```

```
C
10    CONTINUE
C
      END
```

The functions **`parx, pary, parz`** simply prescribe the cartesian coordinates, while the subroutine **`trparv`** provides transformations to more complex domains. In this case, a channel with an innner cylinder around $(0.5, 0.2)$ and radius $r = 0.05$ is prescribed. This file corresponds to the example in chapter 3.

2.2.3. `trigen2d.dat` – parameter file for TRIGEN2D

`M`	FEAT parameter for output $= 0$: no output > 0: output, see FEAT2D manual
`MT`	FEAT parameter for terminal output $= 0$: no output > 0: output, see FEAT2D manual
`ICHECK`	FEAT parameter for subroutine tracing $= 0$: no tracing > 0: tracing, see FEAT2D manual
`IMESH`	parameter for type of parametrization $= 0$: FEAT parametrization $= 1$: OMEGA2D parametrization
`CPARM`	name of OMEGA2D parametrization file
`IBDCHK`	parameter for boundary checking for further refinements $= 0$: no checking > 0: = number of finer boundary points for check of boundary consistency for further refinement
`IBYU`	level for BYU output
`IAVS`	level for AVS output
`NLEV`	number of refined levels-1
`IFMT`	parameter for level of output $= i$: formatted output of mesh data until level i $= -i$: unformatted output of mesh data until level i
`CFILEI`	name of input coarse mesh
`CFILEO`	name of created output coarse mesh
`ITYPEL`	parameter for type of element refinement $= 0$: no adaptive refinement strategy $= 1$: element is partitioned into 9 finer elements for: all elements allowed `DELMA`: distance of element boundary to first refinement line (in percentage) `DELMB`: = no use usual: `DELMA = 0.3333333` $\sim$ equidistant $= 2$: element is partitioned (tangential to domain boundary) into 3 finer stripes for: only elements belonging to 1 (!) boundary component allowed `DELMA`: distance of domain boundary to first refinement line (in percentage) `DELMB`: distance of domain boundary to second refinement line (in percentage) usual: `DELMA = 0.3333333`, `DELMA = 0.6666667` $\sim$ equidistant $= 3$: element is partitioned (tangential to domain boundary) into 2 finer stripes for: only elements belonging to 1 (!) boundary component allowed `DELMA`: distance of domain boundary to first refinement line (in percentage) `DELMB`: no use usual: `DELMA = 0.5` $\sim$ equidistant $= 4$: element is partitioned (normal to domain boundary) into 3 finer stripes for: only elements belonging to 1 (!) boundary component allowed `DELMA`: distance of element boundary to first refinement line (in percentage) `DELMB`: no use usual: `DELMA = 0.3333333` $\sim$ equidistant $= 5$: element is partitioned into 5 finer elements for: all elements allowed `DELMA`: distance of element boundary to first refinement line (in percentage) `DELMB`: no use usual: `DELMA = 0.3333333` $\sim$ equidistant $= 6$: element is partitioned into 3 finer elements for: only elements belonging to 2 (!) boundary component allowed

for: "corner element"
`DELMA`: distance of element boundary to new inner vertex (in percentage)
`DELMB`: no use
usual: `DELMA = 0.5` $\sim$ equidistant

`NELMOD` name of elements to be adaptively refined
list of elements to be refined follows after `DELMB`
one element per line, separated by "return"

`DELMA` parameter for distance in refinement process
$= 0$: no adaptive refinement strategy
$= -i$: unformatted output of mesh data until level i

`DELMB` parameter for distance in refinement process
$= 0$: no adaptive refinement strategy
$= -i$: unformatted output of mesh data until level i

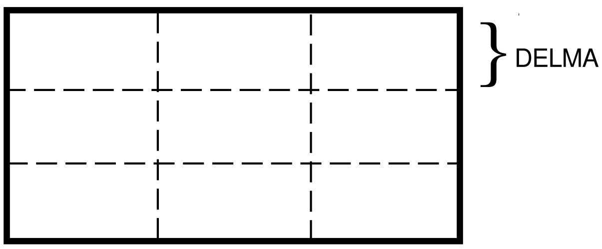

Figure 2.3: `ITYPEL = 1`: Refinement into 9 finer elements

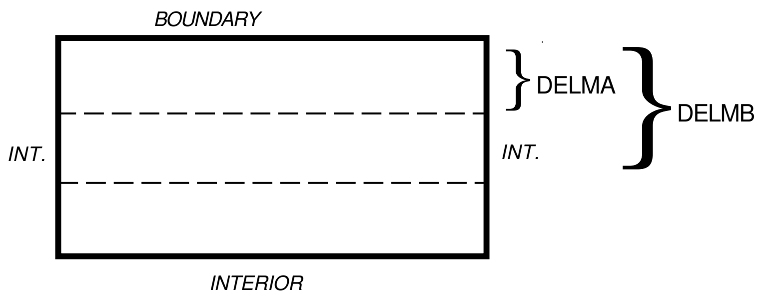

Figure 2.4: `ITYPEL = 2`: Refinement into 3 finer elements

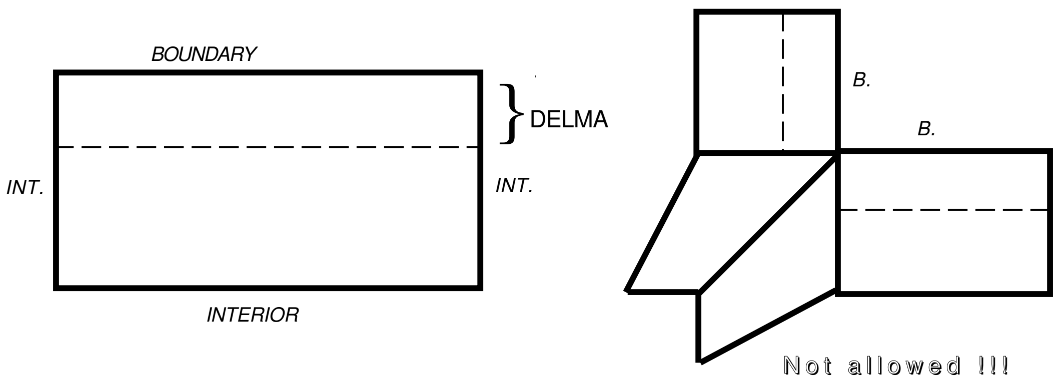

Figure 2.5: `ITYPEL = 3`: Refinement into 2 finer elements

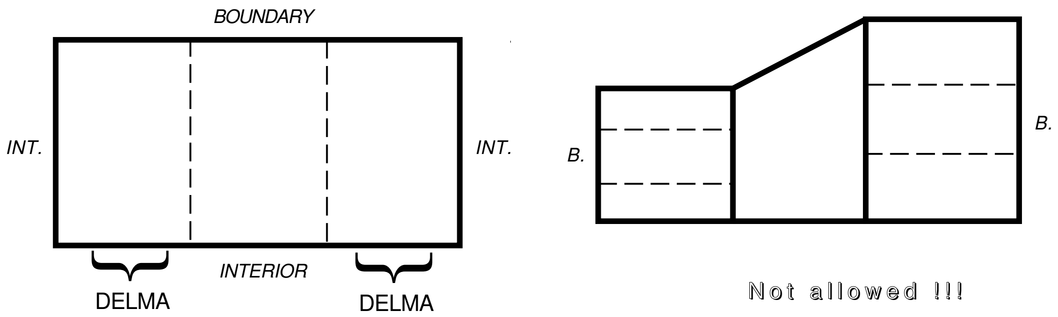

Figure 2.6: `ITYPEL = 4`: Refinement into 3 finer elements

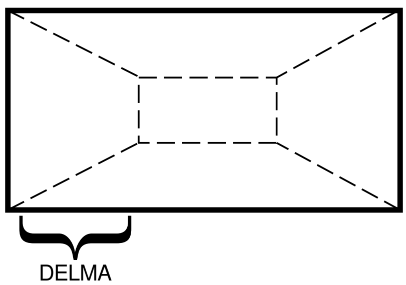

Figure 2.7: `ITYPEL = 5`: Refinement into 5 finer elements

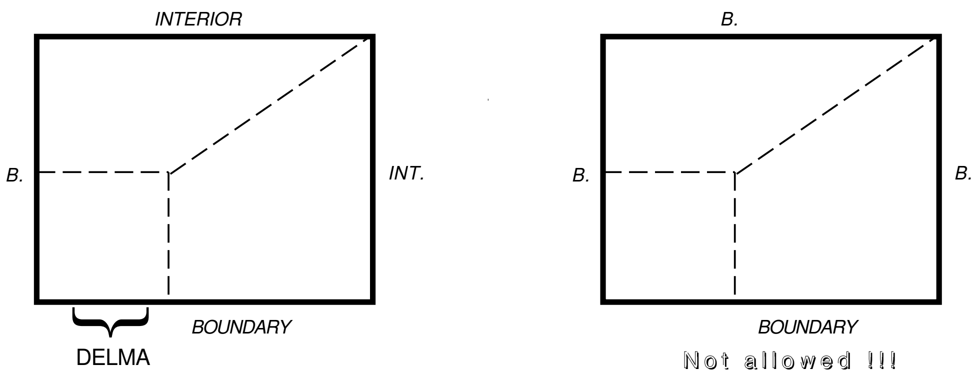

Figure 2.8: `ITYPEL = 6`: Refinement into 3 finer elements

2.2.4. `tr2to3.dat` – parameter file for TR2TO3

`M`	FEAT parameter for output = 0: no output > 0: output, see FEAT2D manual
`MT`	FEAT parameter for terminal output = 0: no output > 0: output, see FEAT2D manual
`ICHECK`	FEAT parameter for subroutine tracing = 0: no tracing > 0: tracing, see FEAT2D manual
`IMESH`	parameter for type of parametrization = 0: FEAT parametrization = 1: OMEGA2D parametrization
`CPARM`	name of OMEGA2D parametrization file
`IBDCHK`	parameter for boundary checking for further refinements = 0: no checking > 0: = number of finer boundary points for check of boundary consistency for further refinement
`IBYU`	level for BYU output
`IAVS`	level for AVS output
`CFILEI`	name of input coarse mesh
`CFILEO`	name of created output coarse mesh
`NPZ`	parameter for number of z-slides list of interior z-coordinates follows after `PZMAX` one element per line, separated by "return"
`PZMIN`	parameter for minimum z-coordinate
`PZMAX`	parameter for maximum z-coordinate

2.2.5. `trigen3d.dat` – parameter file for TRIGEN3D

`M`	FEAT parameter for output
	$= 0$: no output
	> 0: output, see FEAT2D manual
`MT`	FEAT parameter for terminal output
	$= 0$: no output
	> 0: output, see FEAT2D manual
`ICHECK`	FEAT parameter for subroutine tracing
	$= 0$: no tracing
	> 0: tracing, see FEAT2D manual
`IBYU`	level for BYU output
`IAVS`	level for AVS output
`NLEV`	number of refined levels-1
`IFMT`	parameter for level of output
	$= i$: formatted output of mesh data until level i
	$= -i$: unformatted output of mesh data until level i
`CFILEI`	name of input coarse mesh
`CFILEO`	name of created output coarse mesh

2.2.6. `pp2d.dat/pp3d.dat` – parameter files for PP2D and PP3D

The input parameter for the 2D- and the 3D version are almost identical. The parameter files `pp2d.dat/pp3d.dat` and `cc2d.dat/cc3d.dat` differ with respect to a few parameters only.

Specials in `pp2d.dat/pp3d.dat`:

`IMASSL, ICUBML, ISORTU, ICYCU, ILMINU, ILMAXU, IINTU, ISMU, ISLU, NSMU, NSLU, NSMUFA, ISORTP, ICYCP, ILMINP, ILMAXP, IINTP, ISMP, ISLP, NSMP, NSLP, NSMPFA, EPSUR, EPSUD, DMPUD, DMPUMG, DMPUSL, RLXSMU, RLXSLU, AMINU, AMAXU, EPSP, DMPPMG, DMPPSL, RLXSMP, AMINP, AMAXP, IPROJ, PRDIF1, PRDIF2`

`IMESH`	parameter for type of parametrization $= 0$: FEAT parametrization $= 1$: OMEGA2D parametrization (active only in PP2D!!!)
`IRMESH`	input of mesh data $= 0$: create mesh > 0: read mesh (created by TRIGEN2D) > 1: formatted mesh data ($= 1$ unformatted)
`CPARM`	name of OMEGA2D parametrization file
`CMESH`	name of coarse mesh (not longer than 15 characters)
`CFILE`	name of protocol file
`ISTART`	input of start vector $= 0$: start with homogeneous vector (only b.c.'s) $= 1$: read (unformatted) start vector from same level ($= -1$: formatted) $= 2$: read (unformatted) start vector from level-1 ($= -2$: formatted)
`CSTART`	name of start vector file (not longer than 15 characters)
`ISOL`	output of solution vector $= 0$: no output $= 1$: unformatted output on finest level > 1: formatted output on finest level
`CSOL`	name of solution vector file (not longer than 15 characters)
`M`	FEAT parameter for output $= 0$: no output > 0: output, see FEAT2D manual
`MT`	FEAT parameter for terminal output $= 0$: no output > 0: output, see FEAT2D manual
`ICHECK`	FEAT parameter for subroutine tracing $= 0$: no tracing > 0: tracing, see FEAT2D manual
`MSHOW`	parameter for protocol $= 0$: reduced protocol in file `CFILE` $= 1$: expanded protocol in file `CFILE` $= 2$: expanded protocol in file `CFILE` and terminal $= 3$: full protocol in file `CFILE` $= 4$: full protocol in file `CFILE` and terminal
`NLMIN`	parameter for smallest level number
`NLMAX`	parameter for highest level number
`IELT`	parameter for element type

	= 0: E031 parametric
	= 1: E030 parametric
	= 2: E031 nonparametric
	= 3: E030 nonparametric
`ISTOK`	parameter for Stokes calculation
	= 1: Stokes calculation
	$<>$ 1: Navier–Stokes calculation
`IRHS`	parameter for right hand side
	= 0: homogeneous r.h.s
	= 1: steady inhomogeneous r.h.s
	= 2: nonsteady inhomogeneous r.h.s
`IBDR`	parameter for boundary
	= 0: Dirichlet + Neumann boundary values
	= 1: Dirichlet + Neumann + pressure drop boundary values
	= 2: = 1 + time dependent Dirichlet/Neumann conditions
`IERANA`	parameter for error analysis
	= 0: no error analysis
	$>$ 0: error analysis with quadrature formula `IERANA`
`IMASS`	parameter for mass matrix type
	= 0: lumped mass matrix
	= 1: real mass matrix
`IMASSL`	parameter for element type of lumped mass matrix
	= 0: E031 parametric
	= 1: E030 parametric
	= 2: E031 nonparametric
	= 3: E030 nonparametric
`IUPW`	parameter for convective terms
	= 0: streamline diffusion
	= 1: upwinding
`IPRECA`	parameter for diffusive and reactive matrices
	= 0: single precision in RAM
	= 1: double precision in RAM
	= 2: single precision from hard disc
	= 3: double precision from hard disc
	= 4: double precision built up every time
`IPRECB`	parameter for gradient and divergence matrices
	= 0: single precision with usual quadrature, matrix in RAM
	= 1: double precision with usual quadrature, matrix in RAM
	= 2: double precision, exact matrix entries, elementwise application
	= 3: single precision with exact evaluation, matrix in RAM
	= 4: double precision with exact evaluation, matrix in RAM
`ICUBML`	quadrature formula for lumped mass matrix
	$>$ 0: usual lumping
	$<$ 0: diagonal lumping
`ICUBM`	quadrature formula for real mass matrix
`ICUBA`	quadrature formula for Laplacian matrix
`ICUBN`	quadrature formula for convective matrix
`ICUBB`	quadrature formula for gradient matrix
`ICUBF`	quadrature formula for right hand side
`INLMIN`	parameter for smallest number of nonlinear steps matrix
	= 1: linear extrapolation in time (if `INLMAX=1`)
	= -1: constant extrapolation in time (if `INLMAX=-1`)
`INLMAX`	parameter for largest number of nonlinear steps matrix
	= 1: linear extrapolation in time (if `INLMIN=1`)
	= -1: constant extrapolation in time (if `INLMIN=-1`)

`ISORTU`	parameter for renumbering of edges = 1: with respect to x–coordinate = 2: with respect to y–coordinate = 3: with Cuthill-McKee algorithm
`ICYCU`	parameter for mg-cycle for velocity = 0: F–cycle = 1: V–cycle = 2: W–cycle
`ILMINU`	parameter for smallest number of mg-steps for velocity
`ILMAXU`	parameter for largest number of mg-steps for velocity
`IINTU`	parameter for mg-interpolation for velocity = 1: E031 (parametric or nonparametric) = 2: E030 (parametric or nonparametric)
`ISMU`	parameter for mg-smoother for velocity = 1: Jacobi = 2: SOR = 3: SSOR = 4: ILU
`ISLU`	parameter for mg-solver for velocity = 1: SOR = 2: BiCGSTAB = 3: ILU = 4: BiCGSTAB + ILU
`NSMU`	parameter for number of mg-smoothing steps for velocity
`NSLU`	parameter for number of mg-solving steps for velocity
`NSMUFA`	parameter for change of number of mg-smoothing steps on coarser levels for velocity $= n$: $\mathtt{NSMU} * n^{\mathtt{NLMAX}-ILEV}$ smoothing steps if on level $ILEV$
`ISORTP`	parameter for renumbering of elements = 1: with respect to x–coordinate = 2: with respect to y–coordinate = 3: with Cuthill-McKee algorithm
`ICYCP`	parameter for mg-cycle for pressure = 0: F–cycle = 1: V–cycle = 2: W–cycle
`ILMINP`	parameter for smallest number of mg-steps for pressure
`ILMAXP`	parameter for largest number of mg-steps for pressure
`IINTP`	parameter for mg-interpolation for pressure = 1: constant interpolation = 2: rotated trilinear interpolation = 3: trilinear interpolation = 3: modified + optimized rotated trilinear interpolation
`ISMP`	parameter for mg-smoother for pressure = 1: Jacobi = 2: SOR = 3: SSOR = 4: ILU
`ISLP`	parameter for mg-solver for pressure = 1: SOR = 2: CG = 3: ILU = 4: CG + ILU
`NSMP`	parameter for number of mg-smoothing steps for pressure
`NSLP`	parameter for number of mg-solving steps for pressure

NSMPFA	parameter for change of number of mg-smoothing steps on coarser levels for pressure $= n$: $\texttt{NSMP} * n^{\texttt{NLMAX}-ILEV}$ smoothing steps if on level $ILEV$
RE	parameter for viscosity 1/NU
UPSAM	parameter for convectice discretization > 0: upwind: parameter for Samarskij upwinding (usual: UPSAM=1) > 0: SD: leading coefficient with spatial adaptivity (usual: UPSAM=1) < 0: upwind: simple first order upwinding < 0: SD: leading coefficient without spatial adaptivity (usual: UPSAM=-1)
OMGMIN	parameter for lower limit for optimal relaxation in nonlinear iteration > 0: relative changes are calculated < 0: no relative changes are calculated (if OMGMIN = OMGMAX)
OMGMAX	parameter for upper limit for optimal relaxation in nonlinear iteration > 0: relative changes are calculated < 0: no relative changes are calculated (if OMGMIN = OMGMAX)
OMGINI	parameter for start value for calculation of optimal relaxation in nonlinear iteration
EPSUR	stopping criterion for relative changes in velocity in nonlinear iteration
EPSUD	stopping criterion for defect in velocity in nonlinear iteration
DMPUD	stopping criterion for defect improvement in velocity in nonlinear iteration
DMPUMG	stopping criterion for defect improvement in velocity in linear mg-iteration
DMPUSL	stopping criterion for defect improvement for solver in velocity in linear mg-iteration
RLXSMU	relaxation parameter for mg-smoother in velocity
RLXSLU	relaxation parameter for mg-solver in velocity
AMINU	lower limit for optimal correction for mg-solver in velocity
AMAXU	upper limit for optimal correction for mg-solver in velocity
EPSP	stopping criterion for divergence of velocity in pressure equation
DMPPMG	stopping criterion for defect improvement in pressure in linear mg-iteration
DMPPSL	stopping criterion for defect improvement for solver in pressure in linear mg-iteration
RLXSMP	relaxation parameter for mg-smoother in pressure
RLXSLP	relaxation parameter for mg-solver in pressure
AMINP	lower limit for optimal correction for mg-solver in pressure
AMAXP	upper limit for optimal correction for mg-solver in pressure
IPROJ	parameter for type of projection scheme $= 0$: first order (Chorin) > 0: second order (Van Kan) < 0: ABS(IPROJ) steps of first order, then second order
NITNS	maximum number of macro time steps
EPSNS	stopping criterion for time derivative
TIMENS	parameter for absolute start time
THETA	parameter for time stepping value $= 1$: Implicit Euler (first order), if IFRSTP=0 $= 0.5$: Crank–Nicolson (second order), if IFRSTP=0
TSTEP	starting time step
IFRSTP	parameter for time stepping scheme $= 0$: one step schemes $= 1$: fractional step scheme (second order)
INSAV	parameter for unformatted saving of solution vector $= 0$: no saving > 0: macro step size for saving procedure (in #ns)
INSAVN	modulo number of files for unformatted saving of solution vector (maximum = 10)
DTFILM	time difference for unformatted film output
DTAVS	time difference for AVS output
DTBYU	time difference for BYU output
IFUSAV	level for unformatted velocity film output

IFPSAV	level for unformatted pressure film output
IFXSAV	level for unformatted streamline film output active only in PP2D!!!
IAVS	level for AVS output
IBYU	level for BYU output
IFINIT	start file number for film output
IADTIM	parameter for adaptive time step control = 0: no control, fixed time step TSTEP is used = 1: prediction without repetition = 2: prediction with repetition, if nonlinear stopping criteria too large = 3: prediction with repetition, if time error or nonlinear stopping criteria too large < 0: the same as ABS(IADTIM), but with extrapolation in time
TIMEMX	maximum absolute time for stopping
DTMIN	parameter for smallest time step during adaptive control
DTMAX	parameter for largest time step during adaptive control
DTFAC	factor for largest possible time step changes
TIMEIN	absolute time for start procedure
EPSADI	parameter for time error limit in start phase
EPSADL	parameter for time error limit after start phase
EPSADU	upper limit for acceptance for ABS(IADTIM)=3
IEPSAD	parameter for type of error control = 1: control of u(L2) = 2: control of u(MAX) = 3: control of p(L2) = 4: control of p(MAX) = 5: control of MAX(u(L2),p(L2)) = 6: control of MAX(u(MAX),p(MAX)) = 7: control of MAX(u(L2),p(L2),u(MAX),p(MAX)) = 8: control of MIN(u(L2),p(L2),u(MAX),p(MAX))
IADIN	parameter for error control in start phase = 0: EPSADL=EPSADI for T < TIMEIN = 1: EPSADL=linear combination(EPSADI,EPSADL) for T < TIMEIN = 2: EPSADL=logarithmic combination(EPSADI,EPSADL) for T < TIMEIN
IREPIT	maximim number of repetitions for ABS(IADTIM)=3
PRDIF1	parameter for reactive preconditioner (usual: PRDIF1=1)
PRDIF2	parameter for diffusive preconditioner (usual: PRDIF2=0 or 1)

2.2.7. cc2d.dat/cc3d.dat – parameter files for CC2D and CC3D

The input parameter for the 2D- and the 3D version are almost identical. The parameter files `cc2d.dat/cc3d.dat` and `pp2d.dat/pp3d.dat` differ with respect to a few parameters only.

Specials in `cc2d.dat/cc3d.dat`:

IMASSL, ICYCLE, ILMIN, ILMAX, IINT, ISM, ISL, NSM, NSL, NSMFAC, EPSD, EPSDIV, EPSUR, EPSPR, DMPD, DMPMG, EPSMG, DMPSL, EPSSL, RLXSM, RLXSL, AMINMG, AMAXMG, ISTAT

IMESH	parameter for type of parametrization = 0: FEAT parametrization = 1: OMEGA2D parametrization (active only in CC2D!!!)
IRMESH	input of mesh data = 0: create mesh > 0: read mesh (created by TRIGEN2D) > 1: formatted mesh data (= 1 unformatted)
CPARM	name of OMEGA2D parametrization file
CMESH	name of coarse mesh (not longer than 15 characters)
CFILE	name of protocol file
ISTART	input of start vector = 0: start with homogeneous vector (only b.c.'s) = 1: read (unformatted) start vector from same level (= −1: formatted) = 2: read (unformatted) start vector from level-1 (= −2: formatted)
CSTART	name of start vector file (not longer than 15 characters)
ISOL	output of solution vector = 0: no output = 1: unformatted output on finest level > 1: formatted output on finest level
CSOL	name of solution vector file (not longer than 15 characters)
M	FEAT parameter for output = 0: no output > 0: output, see FEAT2D manual
MT	FEAT parameter for terminal output = 0: no output > 0: output, see FEAT2D manual
ICHECK	FEAT parameter for subroutine tracing = 0: no tracing > 0: tracing, see FEAT2D manual
MSHOW	parameter for protocol = 0: reduced protocol in file CFILE = 1: expanded protocol in file CFILE = 2: expanded protocol in file CFILE and terminal = 3: full protocol in file CFILE = 4: full protocol in file CFILE and terminal
NLMIN	parameter for smallest level number
NLMAX	parameter for highest level number
IELT	parameter for element type = 0: E031 parametric = 1: E030 parametric

= 2: E031 nonparametric
= 3: E030 nonparametric
`ISTOK` parameter for Stokes calculation
= 1: Stokes calculation
<> 1: Navier–Stokes calculation
`IRHS` parameter for right hand side
= 0: homogeneous r.h.s
= 1: steady inhomogeneous r.h.s
= 2: nonsteady inhomogeneous r.h.s
`IBDR` parameter for boundary
= 0: Dirichlet + Neumann boundary values
= 1: Dirichlet + Neumann + pressure drop boundary values
= 2: = 1 + time dependent Dirichlet/Neumann conditions
`IERANA` parameter for error analysis
= 0: no error analysis
> 0: error analysis with quadrature formula `IERANA`
`IMASS` parameter for mass matrix type
= 0: lumped mass matrix
= 1: real mass matrix
`IMASSL` parameter for mass matrix lumping
= 0: usual lumping
= 10: diagonal lumping
`IUPW` parameter for convective terms
= 0: streamline diffusion
= 1: upwinding
`IPRECA` parameter for diffusive and reactive matrices
= 0: single precision in RAM
= 1: double precision in RAM
= 2: single precision from hard disc
= 3: double precision from hard disc (not yet)
= 4: double precision built up every time
`IPRECB` parameter for gradient and divergence matrices
= 0: single precision with usual quadrature, matrix in RAM
= 1: double precision with usual quadrature, matrix in RAM (not yet)
= 2: double precision, exact matrix entries, elementwise application (not yet)
= 3: single precision with exact evaluation, matrix in RAM
= 4: double precision with exact evaluation, matrix in RAM (not yet)
`ICUBM` quadrature formula for real mass matrix
`ICUBA` quadrature formula for Laplacian matrix
`ICUBN` quadrature formula for convective matrix
`ICUBB` quadrature formula for gradient matrix
`ICUBF` quadrature formula for right hand side
`INLMIN` parameter for smallest number of nonlinear steps matrix
= 1: linear extrapolation in time (if `INLMAX=1`)
= −1: constant extrapolation in time (if `INLMAX=-1`)
`INLMAX` parameter for largest number of nonlinear steps matrix
= 1: linear extrapolation in time (if `INLMIN=1`)
= −1: constant extrapolation in time (if `INLMIN=-1`)
`ICYCLE` parameter for mg-cycle
= 0: F–cycle
= 1: V–cycle
= 2: W–cycle
`ILMIN` parameter for smallest number of mg-steps
`ILMAX` parameter for largest number of mg-steps
`IINT` parameter for mg-interpolation

	$=1$: rotated trilinear for velocity + constant for pressure
	$=2$: rotated trilinear for velocity + pressure
`ISM`	parameter for mg-smoother
	$=1$: Vanca
`ISL`	parameter for mg-solver
	$=1$: Vanca
`NSM`	parameter for number of mg-smoothing steps
`NSL`	parameter for number of mg-solving steps
`NSMFAC`	parameter for change of number of mg-smoothing steps on coarser levels
	$=n$: $\mathtt{NSM} * n^{\mathtt{NLMAX}-ILEV}$ smoothing steps if on level $ILEV$
`RE`	parameter for viscosity 1/NU
`UPSAM`	parameter for convectice discretization
	>0: upwind: parameter for Samarskij upwinding (usual: `UPSAM=1`)
	>0: SD: leading coefficient with spatial adaptivity (usual: `UPSAM=1`)
	<0: upwind: simple first order upwinding
	<0: SD: leading coefficient without spatial adaptivity (usual: `UPSAM=-1`)
`OMGMIN`	parameter for lower limit for optimal relaxation in nonlinear iteration
	>0: relative changes are calculated
	<0: no relative changes are calculated (if `OMGMIN = OMGMAX`)
`OMGMAX`	parameter for upper limit for optimal relaxation in nonlinear iteration
	>0: relative changes are calculated
	<0: no relative changes are calculated (if `OMGMIN = OMGMAX`)
`OMGINI`	parameter for start value for calculation of optimal relaxation in nonlinear iteration
`EPSD`	stopping criterion for defect in velocity in nonlinear iteration
`EPSDIV`	stopping criterion for defect in divergence in nonlinear iteration
`EPSUR`	stopping criterion for relative changes in velocity in nonlinear iteration
`EPSPR`	stopping criterion for relative changes in pressure in nonlinear iteration
`DMPD`	stopping criterion for defect improvement in nonlinear iteration
`DMPMG`	stopping criterion for defect improvement in linear mg-iteration
`EPSMG`	stopping criterion for defect limit in linear mg-iteration
`DMPSL`	stopping criterion for defect improvement for solver in linear mg-iteration
`EPSSL`	stopping criterion for defect limit for solver in linear mg-iteration
`RLXSM`	relaxation parameter for mg-smoother
`RLXSL`	relaxation parameter for mg-solver
`AMINMG`	lower limit for optimal correction for mg-solver
`AMAXMG`	upper limit for optimal correction for mg-solver
`ISTAT`	parameter for type of problem
	$=0$: steady Navier-Stokes problem
	$=1$: nonsteady Navier-Stokes problem
`NITNS`	maximum number of macro time steps
`EPSNS`	stopping criterion for time derivative
`TIMENS`	parameter for absolute start time
`THETA`	parameter for time stepping value
	$=1$: Implicit Euler (first order), if `IFRSTP=0`
	$=0.5$: Crank–Nicolson (second order), if `IFRSTP=0`
`TSTEP`	starting time step
`IFRSTP`	parameter for time stepping scheme
	$=0$: one step schemes
	$=1$: fractional step scheme (second order)
`INSAV`	parameter for unformatted saving of solution vector
	$=0$: no saving
	>0: macro step size for saving procedure (in `#ns`)
`INSAVN`	modulo number of files for unformatted saving of solution vector (maximum = 10)

`DTFILM`	time difference for unformatted film output
`DTAVS`	time difference for AVS output
`DTBYU`	time difference for BYU output
`IFUSAV`	level for unformatted velocity film output
`IFPSAV`	level for unformatted pressure film output
`IFXSAV`	level for unformatted streamline film output active only in PP2D!!!
`IAVS`	level for AVS output
`IBYU`	level for BYU output
`IFINIT`	start file number for film output
`IADTIM`	parameter for adaptive time step control = 0: no control, fixed time step `TSTEP` is used = 1: prediction without repetition = 2: prediction with repetition, if nonlinear stopping criteria too large = 3: prediction with repetition, if time error or nonlinear stopping criteria too large < 0: the same as `ABS(IADTIM)`, but with extrapolation in time
`TIMEMX`	maximum absolute time for stopping
`DTMIN`	parameter for smallest time step during adaptive control
`DTMAX`	parameter for largest time step during adaptive control
`DTFAC`	factor for largest possible time step changes
`TIMEIN`	absolute time for start procedure
`EPSADI`	parameter for time error limit in start phase
`EPSADL`	parameter for time error limit after start phase
`EPSADU`	upper limit for acceptance for `ABS(IADTIM)`=3
`IEPSAD`	parameter for type of error control = 1: control of u(L2) = 2: control of u(MAX) = 3: control of p(L2) = 4: control of p(MAX) = 5: control of MAX(u(L2),p(L2)) = 6: control of MAX(u(MAX),p(MAX)) = 7: control of MAX(u(L2),p(L2),u(MAX),p(MAX)) = 8: control of MIN(u(L2),p(L2),u(MAX),p(MAX))
`IADIN`	parameter for error control in start phase = 0: `EPSADL`=`EPSADI` for T < `TIMEIN` = 1: `EPSADL`=linear combination(`EPSADI`,`EPSADL`) for T < `TIMEIN` = 2: `EPSADL`=logarithmic combination(`EPSADI`,`EPSADL`) for T < `TIMEIN`
`IREPIT`	maximim number of repetitions for `ABS(IADTIM)`=3

2.3. The file structure of FEATFLOW

In reversed order we arrive at the point to explain the internal structure of FEATFLOW. FEATFLOW consists of 6 directories: `application`, `graphic`, `manual`, `object`, `source` and `utility`. We want to explain their tasks in detail, not following the literal order.

2.3.1. Subdirectory `source` – source code for FEATFLOW

The directory `source` contains (almost) all source code for the preprocessing and solver tools. These are OMEGA2D, TRIGEN2D, TRIGEN3D, TR2TO3, INTPOL2D, INTPOL3D, CC2D, CC3D, PP2D and PP3D. Each of them is split into two, resp., three directories, namely `src` and `dev`, resp., `mg`.

The idea is that `src` contains all information to build up the corresponding system libraries (see `object` and the instructions for installation). These are needed for running an user–application. `mg` is similar and is needed for building the corresponding libraries containing the multigrid components for solving (if necessary). They are only needed by the solvers CC2D, CC3D, PP2D and PP3D.

The directory `dev` is a corresponding directory containing all source- and makefiles, and it is thought to be a "developer directory" which is needed by the experienced user to modify the FEATFLOW software.

Furthermore, the `source` directory contains the code for BLAS, FEAT2D, FEAT3D and OMEGA2D. In an analogous way, `src` contains all files to build up the corresponding system libraries during installation. Additionally, there are two `testdir` directories containing a small test program to get familiar with FEAT. Additionally, we think about adding the GNU FORTRAN77 compiler to provide a FORTRAN compiler for everybody.

2.3.2. Subdirectory `manual` – manuals for FEATFLOW

The directory `manual` contains (almost) all LATEX–source files (in `src`) and postscript–files (in `ps`) for needed manuals. These are OMEGA2D, FEAT2D, FEAT3D and this FEATFLOW manual.

2.3.3. Subdirectory `object` – system software for FEATFLOW

The directory `object` contains the system software of FEATFLOW which is needed for installation, for user–applications, and for generating a (compressed) binary data–file containing the FEATFLOW code. There are three subdirectories, `extract`, `makefiles` and `libraries`.

`extract` is a directory having shell scripts for generating a (compressed) tarfile containing the FEATFLOW package. There are several levels of FEATFLOW data files:

Level 00	complete FEATFLOW (even with all libraries)
Level 01	complete FEATFLOW sources (but without compiled libraries)
Level 02	partial FEATFLOW sources (without `dev` directories)
Level 10	partial FEATFLOW sources + libraries (without multigrid sources)
Level 20	FEATFLOW libraries (without any sources)

The subdirectory `makefiles` contains all makefiles for building up the system libraries and for user–applications. They are available for a class of machines (see the file `MACHINES`) and have to be modified for computers or compilers not belonging to this list. Each of these machine–dependent directories contains some shell scripts, beginning with `make_`, which have to be executed for installation (see the section on installation). In the next versions, there will be some more computers added to our list, containing optimized machine–dependent compiler options.

Finally, the directory `libraries` contains the compiled system libraries which have to be linked to each user–application. The actual FEATFLOW libraries created by the installation (depending on the makefile used) is in the subdirectory `libgen`, while in `libspec` machine- and compiler–dependent system libraries are offered in correspondance to the special makefiles (see above).

2.3.4. Subdirectory `application` – applications under FEATFLOW

`application` is a directory thought to be the place for user–applications. The idea is to use a separate subdirectory for each application. New subdirectories should be generated by linking or copying, at least the `input_files` and the `#data` directory.

FEATFLOW is installed with 5 subdirectories: `user_start`, `comp`, `dat_example`, `example` and `workspace`.

`user_start` is a typical working directory for the user, containing all needed input parameter files in `#data` and a special subdirectory `input_files`. In this directory, the user can find all makefiles, copied during installation and adapted to the machine used, and the input files `parq2d.f`, `parq3d.f` `indat2d.f` and `indat3d.f` (see above). These files have to be edited and modified according to the actual application, and the corresponding makefile has to be executed. Additionally, before compiling, the corresponding storage has to be defined, by defining the corresponding `.inc` file (see above and the later subsection on the `workspace` subdirectory).
After compiling the tool (by executing the makefile) the compiled program should be copied or moved to the parent directory (= `user_start`), and the application may be started there, after editing the corresponding parameter file in `#data`. All other applications should follow these instructions to make life easier.

In a similar way, `comp` is a pre–installed application directory, performing test calculcations (similar to the DFG–benchmark configuration of 1995, see [16]). These applications are thought to provide reference results and CPU times for verifying the installed FEATFLOW version and to obtain benchmark results for different computer types for comparisons. These results can be found in `results`.

`example` is a pre–installed application directory, containing the example programs of Chapter 3.

Finally, there is the directory `workspace` containing some shell scripts. Their only use is to select the corresponding include–files containing the size of storage amount. For instance, if the directory `user_start` is in progress, a link has to be set to the corresponding `.inc` files in the directory `user_start`. This is done by executing the appropriate `links.user_start` file (or other files analogously). This process has to be done before compiling.

2.3.5. Subdirectory `graphic` – graphic tools for FEATFLOW

This directory provides graphic features which are helpful for FEATFLOW. In the present version there are the subdirectories `avs`, `byu` and `gnuplot`. These features will be massively expanded in future versions.

`avs` contains a `.avsrc` file (as link to `avsrc`) and a directory `appl` containing special applications. This `.avsrc` file has to be edited and adapted to the existing configuration. Up to now, AVS has to be executed over network (if available), and the manual has to be taken from there, too.

The subdirectories `byu` and `gnuplot` contain some helpful files for handling the packages MOVIE.BYU or CQUEL.BYU (over network, if available) and for working with GNUPLOT, which may be used for 1D graphics.

2.3.6. Subdirectory `utility` – utilities for FEATFLOW

This directory contains software which can be helpful for the use of FEATFLOW. In the present version there are only programs which provide an a priori–estimate for the needed storage amount. They simply have to be compiled (`f77 file.f -o file`) and to be moved to the directory used for the application, for instance to `application/user_start`. Then, it reads the corresponding parameter file and gives an estimate for the `NNWORK` value needed.

2.4. Installation of FEATFLOW

The usual installation process is the following:

Step 1: Unzip and tar

Create a directory called `featflow` or similar, which will contain all FEATFLOW data. Usually, the user has obtained a binary file `featflow.tar.gz` or `levi_file.tar.gz` ("i" stands for level, see above). This has to be moved to the created directory in which the FEATFLOW tree structure will be installed. It has to be decompressed

```
gunzip featflow.tar.gz
```

and then, a `tar` has to be started

`tar -xvf featflow.tar`.

This generates the complete FEATFLOW tree structure, and the file `featflow.tar` may be removed.

In most cases, all makefiles are already prepared if you work on a 'normal' platform (SUN, IBM, SGI, DEC,, PC+LINUX, etc.) such that the following installation step is very easy:

<u>Step 2a:</u> Automatic installation

Read the README file and execute the installation script:

`install_feat`

After selecting the supported platform, the installation procedure starts! You may continue with Step 3

If you work on a 'nonstandard' platform (whatsoever), you must manually perform the following two steps:

<u>Step 2b:</u> Editing of makefiles and installation scripts

In the next step go to `featflow/object/makefiles`. There are different subdirectories corresponding to various computer types and compiler options. If not, copy one to create your owen, or modify the makefiles in `example`. Let us assume you are in the directory `featflow/object/makefiles/example`. There wait two tasks for you:

1) Edit the shell scripts `make_copy` and `make_lib`, and change the shell variable `FEATFLOW`, containing the right location of FEATFLOW, and the variable `MAKEFILES` where you are.

2) Edit the makefiles and modify, if necessary, the compiler options. These are the lines beginning with `COMOPT =`. This can be done by hand, or by using the shell script `make_change`. Analogously, the shell variable `FEATFLOW` has to be defined correctly as above, using `make_change`.

<u>Step 2c:</u> Installation

Assuming that you are in `featflow/object/makefiles/examples` (or any other directory containing your edited makefiles and shell scripts of step 2). Then, you have to do the following:

0.1) Be sure that you use a C-shell `/bin/csh`.

0.2) Be sure that `featflow/object/libraries/libgen` exists.

0.3) Be sure that the correct `ztime.f` file in `featflow/source/feat2d/src` is taken.

1.0) Execute the shell script `make_copy`.

2.0) Execute the shell script `make_lib`.

This last step takes between 10 and 60 minutes. All libraries will be generated, and all makefiles needed for applications are copied to `featflow/application/user_start`, `featflow/application/example` and `featflow/application/comp`.

Step 3: Test and application

As described in subsection 2.3.4 go to `featflow/application/user_start`, to `featflow /application/example` or to `featflow/application/comp`, and start an application, or create a new directory for your application as described above.

3. Examples for the use of FEATFLOW

This chapter demonstrates explicitely the use of FEATFLOW, performing first the installation, and then applying FEATFLOW for solving a 2D and 3D problem which represents the typical application of a flow in a channel around a cylinder. The values to be computed are lift and drag and pressure differences on the cylinder surface, in a stationary as well as nonstationary configuration. These examples are almost identical with the DFG–benchmark 1995, see [16]), only the 2D case is a little bit different (longer channel!!!).

We show (in short form) how FEATFLOW is manually installed, followed by designing a first coarse mesh with OMEGA2D. This triangulation is refined in a pre–adaptive way by TRIGEN2D, generating the full sequence of nested meshes which are neccessary for the multigrid solvers. We perform a stationary (by CC2D) and a nonstationary calculation (by PP2D), and demonstrate how to interpret the data output and how to use AVS for visualization.

These 2D calculations are followed by a 3D application. We first use TR2TO3 to construct an adequate 3D coarse mesh for this channel configuration. We demonstrate the use of TRIGEN3D to generate the 3D multigrid structure and, finally, we perform the same calculations with CC3D and PP3D as in the 2D case.

3.1. The installation example

We assume that you got the binary file `featflow.tar.gz` and that you created the directory `/home/people/example/featflow` in which you should move this binary file. Being there, you have to perform the following commands:

```
gunzip featflow.tar.gz
tar -xvf featflow.tar
rm featflow.tar.
```

Now, the complete source code of FEATFLOW is installed.

In most cases, all makefiles are already prepared if you work on a ‘normal’ platform (SUN, IBM, SGI, DEC, HP, PC+LINUX, etc.) such that the following installation step is very easy:

Read the README file and execute the installation script:

```
install_feat
```

After selecting the supported platform, the installation procedure starts! You may continue with the 2D and 3D examples!

If you work on a 'nonstandard' platform (whatsoever), you must manually perform the following two steps:

For the following, we assume you have a SUN Sparc 10 (or compatible) with the SUN FORTRAN77 compiler, version 2.0. Next you go to `/home/people/example/featflow/object/makefiles`, and do the following copy:

```
cp -rp sun_sparc10_2.0 my_compiler
cd my_compiler.
```

Edit, for instance, the file `pp2d.m` to figure out how the shell–variables `FEATFLOW` and `COMOPT` are defined. `FEATFLOW` should be set to:

`FEATFLOW=/home/people/example/featflow`,

while `COMOPT` has to be modified for other machines or compilers only. These changes could be done by hand, or by using the shell–script `make_change` which performs (after appropriate replacements) an `sed` command on all files. It is necessary that you use a C-shell `/bin/csh`. Additionally, the variable `MAKEFILES` in `make_lib` should be set appropriate. From now on, we use `FEATFLOW` as abbreviation for `/home/people/example/featflow`.

Before we continue be sure that `FEATFLOW/object/libraries/libgen` exists (if not, create them by the `mkdir` command), and that the "right" subroutine for time measurements are taken. That means, for SUN SPARC 10 (and many others),

```
cd FEATFLOW/source/feat2d/src
cp ztime.sun_sparc ztime.f
cd FEATFLOW/object/makefiles/my_compiler.
```

Now, do the following:

Execute the shell script `make_copy` by typing: `make_copy`.
Execute the shell script `make_lib` by typing: `time make_lib`.

This last step takes between 10 and 60 minutes (the `time` command is not necessary). All FEATFLOW libraries will be generated, and all makefiles needed for applications are copied to `FEATFLOW/application/example` (and to `FEATFLOW/application/user_start` and `FEATFLOW/application/comp`).

Next, go to `FEATFLOW/application/workspace` and run the shell script `links.example` which activates the applications in `FEATFLOW/application/example` (more precise: the `.inc` files in `FEATFLOW/application/example/input_files` will be used for defining the storage amount).

Now, we are ready to start the 2D and 3D examples.

3.2. The 2D example

We start with describing our 2D domain which shall look like:

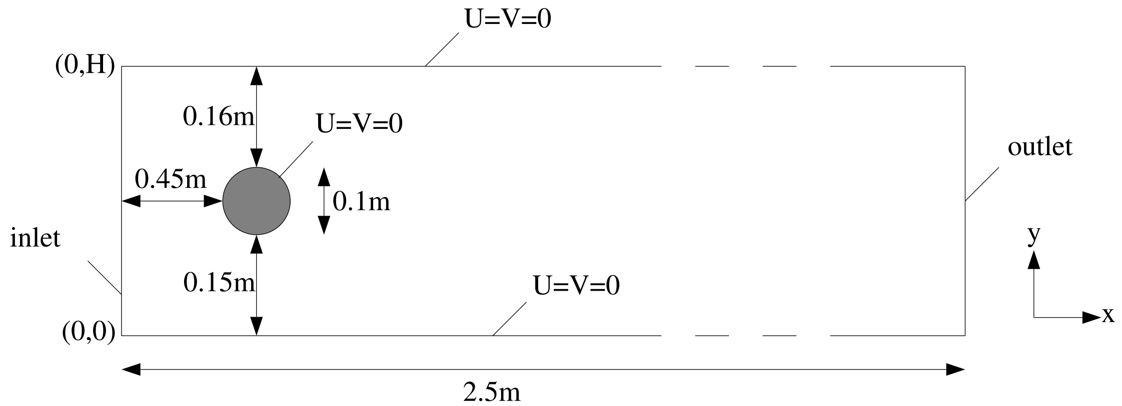

Figure 3.1: 2D domain: channel with cylinder

This can be easily done with OMEGA2D, requiring a computer with PC emulation (at least an AT 286 or higher) and WINDOWS 3.1 (or higher).
We start in OMEGA2D with describing the two boundary components: first, we prescribe the outer rectangle, by setting the first point at the origin, and then proceeding in counter-clockwise sense, until this boundary "curve" is closed. Second, we draw a circle, with starting point at $(0.45, 0.20)$ and center point $(0.50, 0.20)$. The end point is again at $(0.45, 0.20)$, and it is very important to perform a "negative" circle, that means the parametrization is proceeding in a clockwise sense. For more details, look at the OMEGA2D manual.

Next we define mesh points at the boundary components and in the interior. It is remarkable that boundary points cannot be set arbitrarily, but next to another (existing) boundary point, and then they can be moved along the boundary curve. Furthermore, you are always able to "undo" your last actions or to remove points (and even makros and boundary components when needed).

In parallel to defining mesh points, "makros" (that means quadrilateral elements) can be defined, by clicking at four mesh points to form an element. Finally, this session has to be saved (let us assume as `c2d.geb`, `c2d.prm` and `c2d.tri`), and the files `c2d.geb`, `c2d.prm` and `c2d.tri` have to be transferred (by ftp or whatever) to the subdirectory `FEATFLOW/application/example/#pre`, and additionally the coarse mesh `c2d.tri` as `c2d0.tri` to `FEATFLOW/application/example/#adc`. Now, we are ready to proceed with FEATFLOW on our UNIX workstation.

Next, we want to "improve" our coarse mesh `c2d0.tri` to obtain a "better" triangulation.

Therefore, we go to`FEATFLOW/application/example/input_files`, execute the makefiles (ending with `.m`) and move the compiled objects to the parent directory, namely to `FEATFLOW/application/example`. More in detail, for performing the 2D example:

Figure 3.2: Coarsest mesh `c2d0.tri`: 30 elements

```
cd FEATFLOW/application/example/input_files
make -f trigen2d.m ; mv trigen2d ..
cd FEATFLOW/application/example.
```

Next, we want to refine our mesh `c2d0.tri` around and before and behind the circle. This is done by TRIGEN2D. If we perform

```
cp #data/trigen2d.dat_0 #data/trigen2d.dat
time trigen2d
```

we obtain, in `#adc`, a new coarse mesh `c2d1.tri` which is refined in an appropriate way (with `ITYPEL=1`, that means the marked elements are refined into 9 elements). The next steps generate the "final" coarse mesh `c2d2.tri` which is additionally refined around the circle only (with `ITYPEL=3`).

```
cp #data/trigen2d.dat_1 #data/trigen2d.dat
time trigen2d.
```

Figure 3.3: Refined coarse mesh `c2d1.tri`: 130 elements

Figure 3.4: Further refined coarse mesh `c2d2.tri`: 148 elements

Now, we are ready to start a Navier–Stokes calculation. Let us begin with a stationary example, for Reynolds number 20. Doing that, we use the data file `input_files/indat2d.f_stat`, and the following procedure has to be performed

```
cd FEATFLOW/application/example/input_files
cp indat2d.f_stat indat2d.f ; make -f cc2d.m ; mv cc2d ..
cd FEATFLOW/application/example.
```

This data file `input_files/indat2d.f_stat` contains the following definitions:

- The inflow velocity profile at $x = 0.0$ is parabolic, with a maximum value $u_{max} = 0.3$.
- There is one boundary part containing natural boundary conditions (outflow !). This is the boundary segment with $x = 2.5$, and the corresponding parameter values range from `DPARN1=1D0` until `DPARN1=2D0`.
- We show the velocity values at the mesh points `KPU(1)=264` (corresponds to the coordinates $(0.65, 0.20)$) and `KPU(2)=17` ($= (0.85, 0.20)$), the pressure values at `KPP(1)=27` ($= (0.45, 0.20)$), `KPP(2)=30` ($= (0.55, 0.20)$), `KPP(3)=316` ($= (0.50, 0.25)$) and `KPP(4)=17` ($= (0.85, 0.20)$), and the flux between the mesh points `KPX(1)=31` and `KPX(2)=6` (under the circle). The difference of the pressure values for `KPP(1)=27` and `KPP(2)=30` determines a typical pressure difference. Additionally, our runtime protocol will give us the mean pressure over the whole circle (by setting `DPI(1,1)=0D0` and `DPI(2,1)=0D0`), and over half of the circle (by `DPI(2,2)=0.5D0`). Finally, we define parameters for the calculation of drag and lift, where `RHO` is a density parameter, `DIST` a typical length scale, and `UMEAN` is a "mean velocity".

Now, we can execute CC2D with given parameter file `#data/cc2d.dat`. A solution is calculated on level `NLMAX=4`, and the corresponding solution vector is saved as (unformatted) file `#data/#DX4_stat`. The chosen discretization scheme for the convective parts is the streamline–diffusion ansatz, and the stopping criterions are 10^{-3} for the maximum of relative changes. As result we obtain the protocol file `#data/cc2d.stat` which has to be explained in more detail.

<u>Protocol file `#data/cc2d.stat`:</u>

```
--------------------------------------------------------------------------------
         INPUT DATA
--------------------------------------------------------------------------------
 Parametrization file =
 #pre/c2d.prm
 Coarse grid file     =
 #adc/c2d2.tri
 Integer parameters of /IPARM/,etc. :
--------------------------------------------------------------------------------
 minimum mg-level:  NLMIN =    1
 maximum mg-level:  NLMAX =    4
 element type    =    3
 Stokes calculation:  ISTOK =    0
 RHS    generation    =    0
 Boundary generation =    0
 Error evaluation    =    0
 mass evaluation     =    0
```

```
lumped mass eval.   =   0
convective part     =   0
Accuracy for ST     =   4
Accuracy for B      =   3
ICUB mass matrix        =   3
ICUB diff. matrix       =   4
ICUB conv. matrix       =   4
ICUB matrices B1,B2     =   8
ICUB right hand side    =   1
minimum of nonlinear iterations: INLMIN =   1
maximum of nonlinear iterations: INLMAX =   10
type of mg-cycle:  ICYCLE =   0
minimum of linear mg steps :  ILMIN =   1
maximum of linear mg steps :  ILMAX =   5
type of interpolation:  IINT =   2
type of smoother :  ISM =   1
type of solver :  ISL =   1
number of smoothing steps :  NSM =   4
number of solver steps :  NSL =   100
factor sm. steps on coarser lev.:NSMFAC=  2
KPRSM,KPOSM ON LEVEL:   1  32  32
KPRSM,KPOSM ON LEVEL:   2  16  16
KPRSM,KPOSM ON LEVEL:   3  8  8
KPRSM,KPOSM ON LEVEL:   4  4  4
KPRSM,KPOSM ON LEVEL:   5  4  4
KPRSM,KPOSM ON LEVEL:   6  4  4
KPRSM,KPOSM ON LEVEL:   7  4  4
KPRSM,KPOSM ON LEVEL:   8  4  4
KPRSM,KPOSM ON LEVEL:   9  4  4
Real parameters of /RPARM/,etc. :
--------------------------------------------------------------------------------
Viscosity parameter:  1/NU =    1000.00000000000
parameter for Samarskij-upwind:  UPSAM =    0.500000000000000
lower limit for optimal OMEGA: OMGMIN =   0.
upper limit for optimal OMEGA: OMGMAX =     2.0000000000000
start value for optimal OMEGA: OMGINI =     1.0000000000000
limit for U-defects   :        EPSD   =     1.0000000000000D-05
limit for DIV-defects :        EPSDIV =     1.0000000000000D-08
limit for U-changes :          EPSUR  =     1.0000000000000D-03
limit for P-changes :          EPSPR  =     1.0000000000000D-03
defect improvement  :          DMPD   =     1.0000000000000D-01
damping of MG residuals     :  DMPMG  =     1.0000000000000D-01
limit for MG residuals      :  EPSMG  =     1.0000000000000D-01
damping of residuals for solving:  DMPSL =     1.0000000000000D-01
limit of changes for solving:    EPSSL =     1.0000000000000D-01
relaxation for the U-smoother: RLXSM =     1.0000000000000
relaxation for the U-solver :  RLXSL =     1.0000000000000
lower limit optimal MG-ALPHA: AMINMG =   -10.0000000000000
upper limit optimal MG-ALPHA: AMAXMG =    10.0000000000000
Parameters of /NS.../ :
--------------------------------------------------------------------------------
Time dependency          : ISTAT  =   0
Number of time steps     : NITNS  =   9
limit for time derivative: EPSNS  =     1.0000000000000D-05
Total time               : TIMENS =   0.
Theta                    : THETA  =     1.0000000000000
Time step                : TSTEP  =     1.0000000000000D-02
Fractional step          : IFRSTP =   1
Stepsize for nonsteady savings: INSAV =   0
Number of files              : INSAVN =   0
Time step for Film           : DTFILM =   0.
Time step for AVS            : DTAVS =     1.0000000000000
Time step for BYU            : DTBYU =     1.0000000000000
Level for velocity           : IFUSAV =   0
Level for pressure           : IFPSAV =   0
Level for streamlines        : IFXSAV =   0
Level for AVS                : IAVS =   4
Level for BYU                : IBYU =   4
Start file                   : IFINIT =   1
Type of adaptivity           : IADTIM =   -3
```

```
 Max. Time                    : TIMEMX =    100.000000000000
 Min. Timestep                : DTMIN  =     1.0000000000000D-06
 Max. Timestep                : DTMAX  =     1.0000000010000
 Max. Timestep change         : DTFACT =     9.0000000010000
 Time for start procedure     : TIMEIN =    0.50000000000000
 EPS for start procedure      : EPSADI =    0.12500000000000
 EPS for acceptance           : EPSADL =     1.2500000000000D-03
 EPS for not acceptance       : EPSADU =    0.50000000000000
 Acceptance criterion         : IEPSAD =   1
 Start procedure              : IADIN  =   2
 Max.numbers of repetitions   : IREPIT =   3
--------------------------------------------------------------------------------
 ILEV,NVT,NMT,NEL,NVBD:   1  165  313  148  34
 ILEV,NVT,NMT,NEL,NVBD:   2  626  1218  592  68
 ILEV,NVT,NMT,NEL,NVBD:   3  2436  4804  2368  136
 ILEV,NVT,NMT,NEL,NVBD:   4  9608  19080  9472  272

 time for grid generation :     3.6833333298564

 ILEV,NU,NA,NB:  1  313  2089  592
 ILEV,NU,NA,NB:  2  1218  8322  2368
 ILEV,NU,NA,NB:  3  4804  33220  9472
 ILEV,NU,NA,NB:  4  19080  132744  37888


 time for initialization of linear operators :     2.3499999046326


--------------------------------------------------------------------------------
 IT RELU     RELP     DEF-U    DEF-DIV  DEF-TOT  RHONL    OMEGNL   RHOMG
--------------------------------------------------------------------------------
  0                   0.58D-02 0.26D-01 0.26D-01
--------------------------------------------------------------------------------
  1 0.10D+01 0.10D+01 0.34D-02 0.34D-04 0.34D-02 0.13D+00 0.99D+00 0.20D+00
  2 0.41D+00 0.49D+00 0.25D-03 0.85D-05 0.25D-03 0.98D-01 0.10D+01 0.87D-01
  3 0.21D+00 0.14D+00 0.87D-04 0.82D-06 0.87D-04 0.15D+00 0.11D+01 0.24D+00
  4 0.64D-01 0.52D-01 0.20D-04 0.27D-06 0.20D-04 0.17D+00 0.11D+01 0.31D+00
  5 0.13D-01 0.45D-02 0.58D-05 0.44D-07 0.58D-05 0.19D+00 0.11D+01 0.33D+00
  6 0.33D-02 0.19D-02 0.14D-05 0.16D-07 0.14D-05 0.19D+00 0.11D+01 0.29D+00
  7 0.62D-03 0.39D-03 0.33D-06 0.38D-08 0.33D-06 0.20D+00 0.11D+01 0.31D+00
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#   1(   1) TIME= 0.000D+00  NORM(U)= 0.2092401D+00  NORM(P)= 0.4859169D-01
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

P(VELO)  0.20913D-01 0.14274D-02 0.16732D+00 0.14309D-02
P(PRES)  0.12008D+00 0.14282D-01-0.81432D-02 0.28207D-01
I(PRES)  0.47304D-01 0.29039D-01
I(FORCE) 0.53363D+01 0.66688D-02
P(FLUX)  0.39419D-01

 STATISTICS :
 NWORK :        1000000
 IWORKG:        116168
 IWMAXG:        119323
 IWORKI:        685234
 IWMAXI:        685234
 IWORK :        685234
 IWMAX :        704450
--------------------------------------------------------------------------------
 total time :     212.94999999553
 appr. time :     212.83334153146
 grid  time :     3.6833333298564
 post  time :     7.5666656494141
 lin.  time :     49.299993515015
 -> mavec time :     6.1333212852478
 -> konv. time :     39.716608047485
 -> bdry  time :     4.9992084503174D-02
 -> LC    time :     3.4000720977783
 mg    time :     152.28334903717
 #substeps  :   1
```

```
 #nonlinear :   7
 #mg        :   14
--------------------------------------------------------------------------------

  MULTIGRID COMPONENTS [in percent]:
  smoothing     :    74.324147510282
  solver        :    6.0085270812218
  defect calc.  :    8.7993897069504
  prolongation  :    9.7515820090053
  restriction   :    1.0944625011425
--------------------------------------------------------------------------------
```

Most statement have not to be explained, only a few ones:

- `ILEV,NVT,NMT,NEL,NVBD` means number of vertices/midpoints/elements/boundary points on level `ILEV`
- `ILEV,NU,NA,NB` means number of midpoints and nonzero matrix entries for the Laplacian/gradient matrix on level `ILEV`
- `RHONL` convergence rate of nonlinear iteration
- `OMEGNL` optimally chosen relaxation parameter for nonlinear iteration
- `RHOMG` multigrid convergence rate for Oseen equation
- `P(VELO)` contains the u- and v–velocity for both grid points defined in `input_files /indat2d.f`.
- `P(PRES)` contains the 4 pressure values for the grid points defined before.
- `I(PRES)` contains the 2 integral mean pressure values defined before.
- `I(FORCE)` contains the drag and lift values defined before.
- `P(FLUX)` contains the flux value defined before.
- `IWMAX` contains the value for `NNWORK` needed. `NWORK` shows the actually defined parameter.

Additionally, we obtain files for graphical output, namely in `#avs` the file `u1.inp`, and in `#byu` the BYU files `u1.vec` (velocity), `p1.scl` (pressure) and `x1.vec` (streamfunction). A typical example for a AVS isoline plot of the pressure can be found in the following picture.

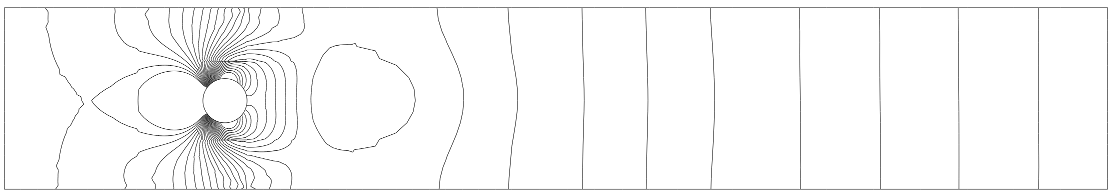

Figure 3.5: Pressure isolines for the stationary 2D calculation

Next, we perform a nonstationary calculation, for Reynolds number about 100. We use the data file `input_files/indat2d.f_non`, and the following procedure has to be performed

```
cd FEATFLOW/application/example/input_files
cp indat2d.f_non indat2d.f ; make -f pp2d.m ; mv pp2d ..
cd FEATFLOW/application/example.
```

This data file `input_files/indat2d.f_non` is almost identical to the preceeding one:

- The inflow velocity profile at $x = 0.0$ is again parabolic, but with a maximum value $u_{max} = 1.5$.
- `UMEAN` is changed for defining drag and lift.

Now, we can execute PP2D with given parameter file `#data/pp2d.dat`. A solution is calculated on level `NLMAX=4`, with (stationary) start solution file `#data/#DX4_stat`. Now, the chosen discretization scheme for the convective parts is the upwinding ansatz, and we perform our time stepping with the fractional step scheme and fully adaptive time step control until `TIMEMX=4D0`. Files for graphical output, namely for `#avs` and `#byu`, are written out all 1 "second". As result we obtain the protocol file `#data/pp2d.non` which is very similar to `#data/cc2d.stat`, but containing information for all time steps performed. We explain this protocol file for only one macro time step.

Protocol file `#data/pp2d.non`:

$\vdots$

```
ILEV,NVT,NMT,NEL,NVBD:   1  165   313   148   34
ILEV,NVT,NMT,NEL,NVBD:   2  626   1218   592   68
ILEV,NVT,NMT,NEL,NVBD:   3  2436   4804   2368   136
ILEV,NVT,NMT,NEL,NVBD:   4  9608   19080   9472   272

time for grid generation :      3.6333331540227

ILEV,NU,NA,NB:  1  313  2089  592
ILEV,NU,NA,NB:  2  1218  8322  2368
ILEV,NU,NA,NB:  3  4804  33220  9472
ILEV,NU,NA,NB:  4  19080  132744  37888

ILEV,NP,NC:  1  148  706
ILEV,NP,NC:  2  592  2892
ILEV,NP,NC:  3  2368  11704
ILEV,NP,NC:  4  9472  47088
```

$\vdots$

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
MACRO STEP   68 AT TIME =  0.294D+01 WITH 1 STEP : DT1 =  0.288D-01
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
--------------------------------------------------------------------------------
 IT REL-U1   REL-U2   DEF-U1   DEF-U2   RHONL    OMEGNL   RHOMG1   RHOMG2
```

```
--------------------------------------------------------------------------------
  0                  0.17D-03 0.23D-03
--------------------------------------------------------------------------------
  1 0.20D+00 0.26D+00 0.14D-04 0.77D-05 0.62D-01 0.10D+01 0.18D-01 0.13D-01
  2 0.45D-01 0.28D-01 0.11D-05 0.14D-05 0.78D-01 0.10D+01 0.37D-01 0.57D-01
--------------------------------------------------------------------------------
  IT DIV-L2   RHOMGP
--------------------------------------------------------------------------------
   6 0.11D-12 0.37D-01
--------------------------------------------------------------------------------
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#   1(  68) TIME= 0.297D+01 REL2(P)= 0.705D+00 RELM(P)= 0.221D+01
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
MACRO STEP   68 AT TIME =  0.294D+01 WITH 3 STEPS: DT3 =  0.960D-02
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
--------------------------------------------------------------------------------
 IT REL-U1   REL-U2   DEF-U1   DEF-U2   RHONL    OMEGNL   RHOMG1   RHOMG2
--------------------------------------------------------------------------------
  0                  0.49D-04 0.67D-04
--------------------------------------------------------------------------------
  1 0.67D-01 0.78D-01 0.15D-05 0.73D-06 0.23D-01 0.10D+01 0.51D-02 0.40D-02
--------------------------------------------------------------------------------
  IT DIV-L2   RHOMGP
--------------------------------------------------------------------------------
   5 0.39D-12 0.35D-01
--------------------------------------------------------------------------------
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#   1(  68) TIME= 0.295D+01 REL2(P)= 0.151D+01 RELM(P)= 0.581D+01
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

--------------------------------------------------------------------------------
 IT REL-U1   REL-U2   DEF-U1   DEF-U2   RHONL    OMEGNL   RHOMG1   RHOMG2
--------------------------------------------------------------------------------
  0                  0.69D-04 0.94D-04
--------------------------------------------------------------------------------
  1 0.93D-01 0.12D+00 0.21D-05 0.10D-05 0.23D-01 0.10D+01 0.52D-02 0.41D-02
  2 0.72D-02 0.62D-02 0.66D-07 0.80D-07 0.29D-01 0.10D+01 0.13D-01 0.26D-01
--------------------------------------------------------------------------------
  IT DIV-L2   RHOMGP
--------------------------------------------------------------------------------
   5 0.31D-12 0.33D-01
--------------------------------------------------------------------------------
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#   2(  68) TIME= 0.296D+01 REL2(P)= 0.153D+01 RELM(P)= 0.537D+01
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

--------------------------------------------------------------------------------
 IT REL-U1   REL-U2   DEF-U1   DEF-U2   RHONL    OMEGNL   RHOMG1   RHOMG2
--------------------------------------------------------------------------------
  0                  0.50D-04 0.66D-04
--------------------------------------------------------------------------------
  1 0.68D-01 0.83D-01 0.15D-05 0.68D-06 0.22D-01 0.10D+01 0.58D-02 0.45D-02
--------------------------------------------------------------------------------
  IT DIV-L2   RHOMGP
--------------------------------------------------------------------------------
   5 0.31D-12 0.35D-01
--------------------------------------------------------------------------------
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#   3(  68) TIME= 0.297D+01 REL2(P)= 0.279D+01 RELM(P)= 0.961D+01
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
OLD DT= 0.96D-02 U(L2)= 0.13D-02 U(MX)= 0.51D-02 P(L2)= 0.45D-02 P(MX)= 0.95D-02
CHOICE  0(  68)    ---- NEW DT =  0.954D-02 -- OLD DT =  0.960D-02
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

P(VELO) -0.12490D-01 0.45097D+00 0.10542D+01-0.67446D+00
P(PRES)  0.17141D+01-0.41863D+00-0.64658D+00 0.17289D+00
```

```
I(PRES)  0.30836D+00-0.41448D-02
I(FORCE) 0.29159D+01-0.54744D+00
P(FLUX)  0.20401D+00
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
#  68 ( 272) TIME= 0.297D+01 RELU(L2)= 0.25D+01 RELP(L2)= 0.20D+01 REL= 0.25D+01
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

 total time :     3364.2498697899
 mavec time :     309.36813378334
 konv. time :     406.31425285339
 bdry  time :     2.0668334960938
 LC    time :     87.866734504700
 ILU   time :     323.13556003571
 U-mg  time :     1063.4651870728
 P-mg  time :     1141.2314567566
```

⋮

```
 STATISTICS :
 NWORK :        1300000
 IWORKG:        116168
 IWMAXG:        119323
 IWORKI:        967903
 IWMAXI:        967903
 IWORK :        1039351
 IWMAX :        1203452
--------------------------------------------------------------------------------
 total time :     5327.6336588524
 appr. time :     5320.4342214540
 grid  time :     3.6333331540227
 post  time :     38.716011047363
 lin.  time :     1797.2014658451
 -> mavec time :     487.26168847084
 -> konv. time :     647.08451652527
 -> bdry  time :     3.0346069335938
 -> LC    time :     139.76541614532
 -> ILU   time :     520.05523777008
 U-mg  time :     1709.3506851196
 P-mg  time :     1771.5327262878
 #substeps  :   428
 #mg P      :   2326
 #nonlinear :   604
 #mg U      :   1220
--------------------------------------------------------------------------------

  U-MULTIGRID COMPONENTS [in percent]:
  smoothing     :    42.218687909096
  solver        :    2.2564278443300
  defect calc.  :    32.417565448349
  prolongation  :    20.300764952555
  restriction   :    2.6163307795102

  P-MULTIGRID COMPONENTS [in percent]:
  smoothing     :    50.378172496605
  solver        :    4.9000111811739
  defect calc.  :    16.941610100891
  prolongation  :    19.692759428133
  restriction   :    7.8014891041156
--------------------------------------------------------------------------------
```

Some explanations:

- `ILEV,NVT,NMT,NEL,NVBD`, `ILEV,NU,NA,NB` as before
- `ILEV,NP,NC` means number of pressure unknowns and nonzero matrix entries for pressure matrix
- We perform macro time step 68, at absolute time $T = 2.94s$, and the actually chosen time step is $t = 0.0096$. As predictor step we perform a calculation with a three times larger time step `DT1=0.288D-01`, and then three substeps with the actual time step size, `DT3=0.960D-02`. Anyway, we perform first the nonlinear iteration for the Burgers–step, which multigrid convergence rates `RHOMG1` and `RHOMG2` for each convection–diffusion equation. Then, the corresponding pressure equation is solved, with multigrid convergence rate `RHOMGP`.
- The estimated time error in different norms (L^2 and L^∞ for velocity and pressure) are written out, followed by the new adaptively chosen time step value.
- `RELU(L2)=0.25D+01` and `RELP(L2)=0.20D+01` are measures for the time derivative of the velocity, resp., the pressure.

Additionally, the point values for `P(VELO)`, ..., `P(FLUX)` are printed separately into corresponding files in the subdirectory **`#points`** and can be visualized by GNUPLOT, for instance. Furthermore, we obtain other files for graphical output, namely in **`#avs`** and in **`#byu`**, written with a "1 second" delay. In the following picture, the corresponding pressure plot for $T = 4$ (file **`#avs/u91.inp`**) is shown.

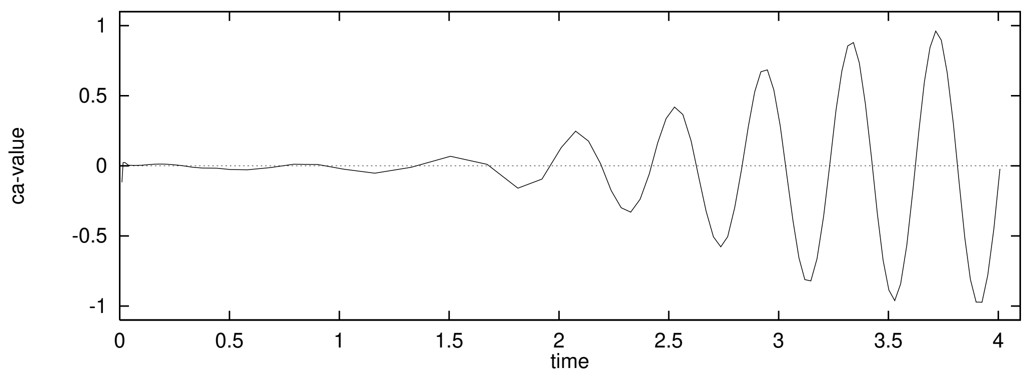

Figure 3.6: Lift values for the nonstationary 2D calculation

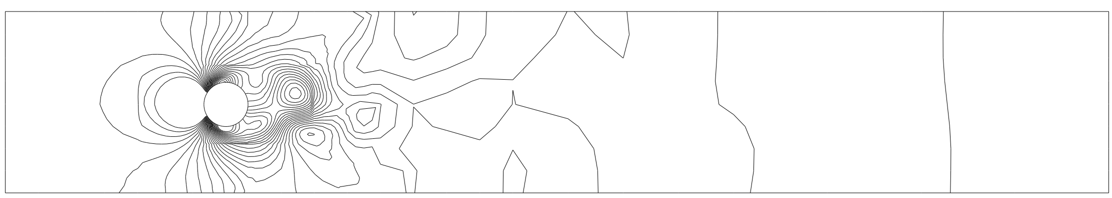

Figure 3.7: Pressure isolines for the nonstationary 2D calculation for $T = 4$

3.3. The 3D example

In this example we want to perform similar calculations in the following 3D domain:

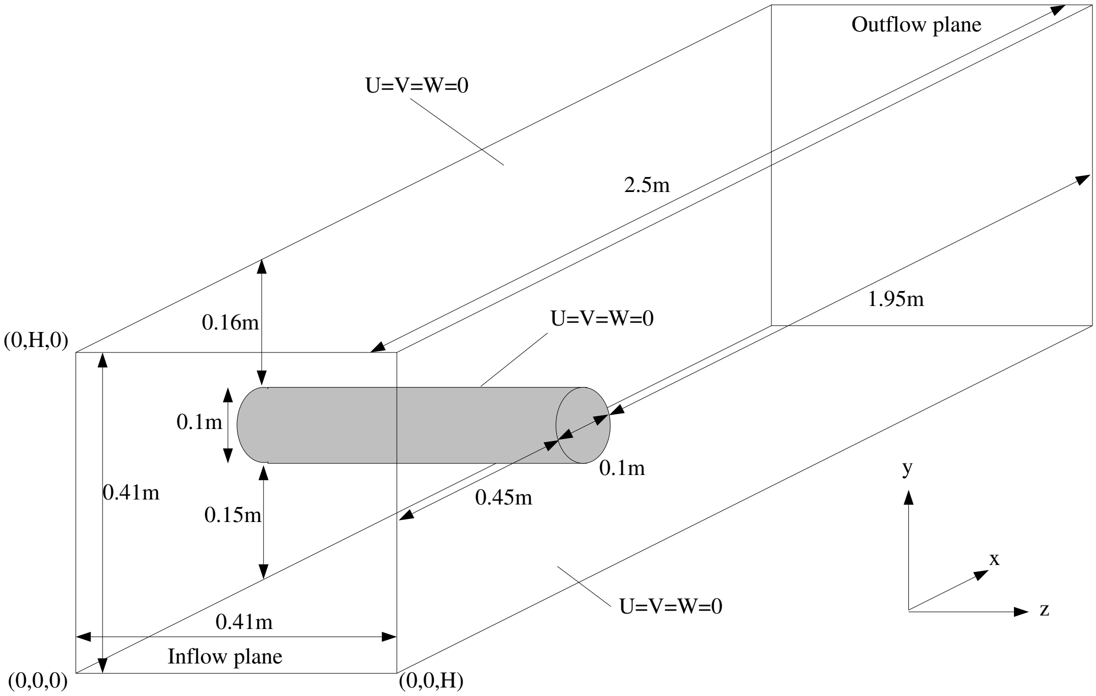

Figure 3.8: 3D domain: channel with cylinder

A corresponding coarse mesh, which is simply a 3D extension of our `c2d2.tri` grid, can be obtained by using TR2TO3. First of all, do the following:

```
cd FEATFLOW/application/example/input_files
make -f tr2to3.m ; mv tr2to3 ..
make -f trigen3d.m ; mv trigen3d ..
cd FEATFLOW/application/example.
```

Next, the corresponding 3D coarse mesh `#adc/c3d2.tri` (see `#data/tr2to3.dat`) is generated by executing TR2TO3. Here, in z–direction, 8 non–equidistant layers are defined. The following AVS picture can be generated by performing TRIGEN3D.

Now, we are ready to start our 3D Navier–Stokes calculation. Again, let us begin with the stationary example, for Reynolds number 20. In this case, we use the data file `input_files /indat3d.f_stat`, and the following, already well known procedure has to be performed

```
cd FEATFLOW/application/example/input_files
cp indat3d.f_stat indat3d.f ; make -f cc3d.m ; mv cc3d ..
cd FEATFLOW/application/example.
```

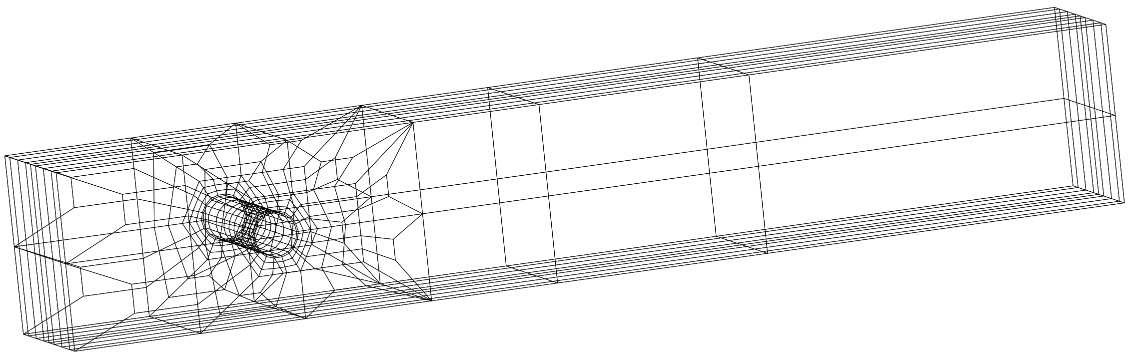

Figure 3.9: Coarse mesh `c3d2.tri`

This data file `input_files/indat3d.f_stat` contains the following definitions:

- The inflow velocity profile at $x = 0.0$ is parabolic, with a maximum value $u_{max} = 0.45$, but leading to the same mean velocity as in the 2D case.
- There is one boundary part containing natural boundary conditions (outflow !). This is the boundary segment with $x = 2.5$, where the corresponding parameter `IFLAG` is activated in subroutine `NEUDAT`.
- The integral mean pressure is calculated over the full and half of the circular cylinder.
- We define similar parameters for the calculation of drag and lift as in 2D, now over the full cylinder surface. `RHO` is a density parameter, `DIST` a typical length scale, and `UMEAN` is a "mean velocity".
- We show the velocity values at the mesh points `KPU(1)=3421` (corresponds to the coordinates $(0.65, 0.20, 0.205)$) and `KPU(2)=677` ($= (0.85, 0.20, 0.205)$), and the pressure values at `KPP(1)=687` ($= (0.45, 0.20, 0.205)$), `KPP(2)=690` ($= (0.55, 0.20, 0.205)$), `KPP(3)=3498` ($= (0.50, 0.25, 0.205)$) and `KPP(4)=677` ($= (0.85, 0.20, 0.205)$). The difference of the pressure values for `KPP(1)=687` and `KPP(2)=690` can be used again for determing a typical pressure difference on the cylinder.

Now, we can execute CC3D with given parameter file `#data/cc3d.dat`. A solution is calculated on level `NLMAX=3`, and the corresponding solution vector is saved as (unformatted) file `#data/#DX3_stat`. The chosen discretization scheme for the convective parts is the streamline–diffusion ansatz, and the stopping criterions are $1 \cdot 10^{-2}$ for the maximum of relative changes. It is remarkable, that the number of smoothing steps `NSM=64` is surprisingly large, due to the chosen anisotropic mesh in combination with the streamline–diffusion method. As result of CC3D we obtain the protocol file `#data/cc3d.stat` which is almost identical to the 2D–version `#data/cc2d.stat`.

Protocol file `#data/cc3d.stat`:

```
--------------------------------------------------------------------------------
         INPUT DATA
--------------------------------------------------------------------------------
 Parametrization file =
 #pre/c3d.prm
 Coarse grid file     =
 #adc/c3d2.tri
 Integer parameters of /IPARM/,etc. :
--------------------------------------------------------------------------------
 minimum mg-level:  NLMIN =   1
 maximum mg-level:  NLMAX =   3
 element type   =   3
 Stokes calculation:  ISTOK =   0
 RHS    generation   =   0
 Boundary generation =   0
 Error evaluation    =   0
 mass evaluation     =   0
 lumped mass eval.   =   0
 convective part     =   0
 Accuracy for ST     =   4
 Accuracy for B      =   3
 ICUB mass matrix        =   7
 ICUB diff. matrix       =   7
 ICUB conv. matrix       =   7
 ICUB matrices B1,B2     =   7
 ICUB right hand side    =   1
 minimum of nonlinear iterations: INLMIN =   1
 maximum of nonlinear iterations: INLMAX =   20
 type of mg-cycle:  ICYCLE =   0
 minimum of linear mg steps :  ILMIN =   1
 maximum of linear mg steps :  ILMAX =   5
 type of interpolation:  IINT =   2
 type of smoother :  ISM =   1
 type of solver :  ISL =   1
 number of smoothing steps :  NSM =   64
 number of solver steps :  NSL =   500
 factor sm. steps on coarser lev.:NSMFAC=  8
 KPRSM,KPOSM ON LEVEL:   1  4096  4096
 KPRSM,KPOSM ON LEVEL:   2  512  512
 KPRSM,KPOSM ON LEVEL:   3  64  64
 KPRSM,KPOSM ON LEVEL:   4  64  64
 KPRSM,KPOSM ON LEVEL:   5  64  64
 KPRSM,KPOSM ON LEVEL:   6  64  64
 KPRSM,KPOSM ON LEVEL:   7  64  64
 KPRSM,KPOSM ON LEVEL:   8  64  64
 KPRSM,KPOSM ON LEVEL:   9  64  64
 Real parameters of /RPARM/,etc. :
--------------------------------------------------------------------------------
 Viscosity parameter:  1/NU =    1000.00000000000
 parameter for Samarskij-upwind:  UPSAM =     1.0000000000000
 lower limit for optimal OMEGA: OMGMIN =   0.
 upper limit for optimal OMEGA: OMGMAX =     1.0000000000000
 start value for optimal OMEGA: OMGINI =     1.0000000000000
 limit for U-defects   :        EPSD   =     1.0000000000000D-05
 limit for DIV-defects :        EPSDIV =     1.0000000000000D-05
 limit for U-changes :          EPSUR  =     5.0000000000000D-02
 limit for P-changes :          EPSPR  =     5.0000000000000D-02
 defect improvement  :          DMPD   =     1.0000000000000D-01
 damping of MG residuals     :  DMPMG  =    0.50000000000000
 limit for MG residuals      :  EPSMG  =    0.50000000000000
 damping of residuals for solving:  DMPSL =     1.0000000000000D-01
 limit of changes for solving:    EPSSL =     1.0000000000000D-01
 relaxation for the U-smoother: RLXSM =     1.0000000000000
 relaxation for the U-solver :  RLXSL =    0.80000000000000
 lower limit optimal MG-ALPHA: AMINMG =   -10.0000000000000
 upper limit optimal MG-ALPHA: AMAXMG =    10.0000000000000
 Parameters of /NS.../ :
```

```
--------------------------------------------------------------------------------
 Time dependency          : ISTAT  =   0
 Number of time steps     : NITNS  =   50
 limit for time derivative: EPSNS  =     1.0000000000000D-05
 Total time               : TIMENS =   0.
 Theta                    : THETA  =     1.0000000000000
 Time step                : TSTEP  =     1.0000000000000D-02
 Fractional step          : IFRSTP =   1
 Stepsize for nonsteady savings: INSAV =   0
 Number of files               : INSAVN =   0
 Time step for Film            : DTFILM =   0.
 Time step for AVS             : DTAVS =     1.0000000000000
 Time step for BYU             : DTBYU =     1.0000000000000
 Level for velocity            : IFUSAV =   0
 Level for pressure            : IFPSAV =   0
 Level for streamlines         : IFXSAV =   0
 Level for AVS                 : IAVS =   3
 Level for BYU                 : IBYU =   2
 Start file                    : IFINIT =   1
 Type of adaptivity            : IADTIM =   -1
 Max. Time                     : TIMEMX =     5.0000000000000
 Min. Timestep                 : DTMIN  =     1.0000000000000D-06
 Max. Timestep                 : DTMAX  =     9.0000000010000
 Max. Timestep change          : DTFACT =     9.0000000010000
 Time for start procedure      : TIMEIN =     1.0000000000000
 EPS for start procedure       : EPSADI =    0.12500000000000
 EPS for acceptance            : EPSADL =     1.2500000000000D-03
 EPS for not acceptance        : EPSADU =    0.50000000000000
 Acceptance criterion          : IEPSAD =   3
 Start procedure               : IADIN  =   2
 Max.numbers of repetitions    : IREPIT =   1
--------------------------------------------------------------------------------
 ILEV,NVT,NAT,NEL,NEL0,NEL1,NEL2:  1  1485  3836  1184  1120  0  64
 ILEV,NVT,NAT,NEL,NEL0,NEL1,NEL2:  2  10642  29552  9472  8864  96  512
 ILEV,NVT,NAT,NEL,NEL0,NEL1,NEL2:  3  80388  231872  75776  70512  1168  4096

 time for grid initialization :     36.883334092796

 ILEV,NU,NA,NB:  1  3836  39356  7104
 ILEV,NU,NA,NB:  2  29552  313712  56832
 ILEV,NU,NA,NB:  3  231872  2505152  454656


 time for initialization of linear operators :     62.316661834717


--------------------------------------------------------------------------------
 IT RELU     RELP     DEF-U    DEF-DIV  DEF-TOT  RHONL    OMEGNL   RHOMG
--------------------------------------------------------------------------------
  0                   0.26D-03 0.68D-03 0.72D-03
--------------------------------------------------------------------------------
  1 0.10D+01 0.10D+01 0.93D-04 0.48D-05 0.93D-04 0.13D+00 0.10D+01 0.12D+00
  2 0.51D+00 0.42D+00 0.14D-03 0.84D-06 0.14D-03 0.43D+00 0.10D+01 0.18D+00
  3 0.26D+00 0.26D+00 0.28D-04 0.11D-06 0.28D-04 0.34D+00 0.10D+01 0.77D-01
  4 0.10D+00 0.39D-01 0.34D-05 0.20D-06 0.34D-05 0.26D+00 0.99D+00 0.22D+00
  5 0.40D-01 0.31D-01 0.29D-05 0.79D-07 0.29D-05 0.33D+00 0.99D+00 0.69D+00
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#   1(   1) TIME= 0.000D+00  NORM(U)= 0.2160847D+00  NORM(P)= 0.7659621D-01
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

P(VELO)  0.32444D-01 0.12893D-02 0.21419D-04 0.20393D+00 0.16592D-02-0.28822D-04
P(PRES)  0.19791D+00 0.36067D-01 0.19174D-02 0.45314D-01
I(PRES)  0.61808D-01 0.95865D-01
I(FORCE) 0.63585D+01 0.38097D-02

 STATISTICS :
 NWORK :         10000000
 IWORKG:         1339516
 IWMAXG:         3450964
 IWORKI:         9732825
```

```
 IWMAXI:        9732825
 IWORK :        9732825
 IWMAX :        9893601
--------------------------------------------------------------------------------
 total time :     21627.316276040
 appr. time :     21626.999564104
 grid  time :     36.883334092796
 post  time :     51.515625000000
 lin.  time :     1290.7828254700
 -> mavec time :     109.70226669312
 -> konv. time :     1106.0506057739
 -> bdry  time :    0.75054168701172
 -> LC    time :     74.279411315918
 mg    time :     20247.817779541
 #substeps  :   1
 #nonlinear :   5
 #mg        :   6
--------------------------------------------------------------------------------

  MULTIGRID COMPONENTS [in percent]:
  smoothing     :    98.178536576809
  solver        :   0.46786725951263
  defect calc.  :   0.70117728389642
  prolongation  :   0.59852068627434
  restriction   :    5.3898193507251D-02
--------------------------------------------------------------------------------
```

Only a few differences with respect to the 2D example have to be explained:

– `ILEV,NEL0,NEL1,NEL2` appears in combination with streamline–diffusion only. It presents the number of element requiring fully trilinear/linear/axiparallel transformations on the reference element.

– `P(VELO)`, ..., `I(IFORCE)` are analogously defined as in the 2D case.

As before, we obtain files for graphical output, namely in `#avs` the file `u1.inp`, and in `#byu` the BYU files `u1.vec` (velocity) and `p1.scl` (pressure). A typical example for a AVS velocity plot in the midplane is shown in the following picture.

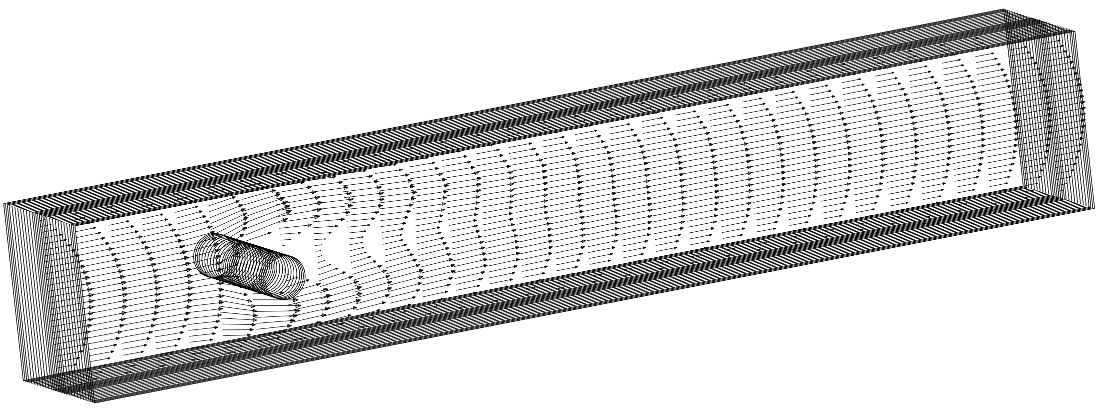

Figure 3.10: Velocity plot in the midplane $z = 0.205$ for the stationary 3D calculation

Next, we perform a nonstationary calculation, again for Reynolds number 100. We use the data file `input_files/indat3d.f_non`, and the same "well known" procedure has to be performed:

```
cd FEATFLOW/application/example/input_files
cp indat3d.f_non indat3d.f ; make -f pp3d.m ; mv pp3d ..
cd FEATFLOW/application/example.
```

This data file `input_files/indat3d.f_non` is almost identical to the stationary one:

- The inflow velocity profile at $x = 0.0$ is again parabolic, but with a maximum value $u_{max} = 2.25$.
- `UMEAN` is changed for defining drag and lift.

Now, we can execute PP3D with given parameter file `#data/pp3d.dat`. A solution is calculated on level `NLMAX=3`, with (stationary) start solution file `#data/#DX3_stat`. Now, the chosen discretization scheme for the convective parts is done by upwinding, and we perform our time stepping with the fractional step scheme and fully adaptive time step control until `TIMEMX=9D0`. Files for graphical output, namely for `#avs` and `#byu`, are written out all 1 "second". We obtain the protocol file `#data/pp3d.non` which is almost identical to the preceeding protocol files. Therefore, we renounce a more precise explanation.

Again, the point values for `P(VELO)`, ..., `I(FORCE)` are printed separately into corresponding files in the subdirectory `#points` and be visualized by GNUPLOT, for instance (see Figure 3.6). Furthermore, we obtain other files for graphical output, namely in `#avs` and in `#byu`, written with a "1 second" delay. In the following Figure 3.12, the corresponding velocity plot for $T = 9$ (file `#avs/u110.inp`) is shown.

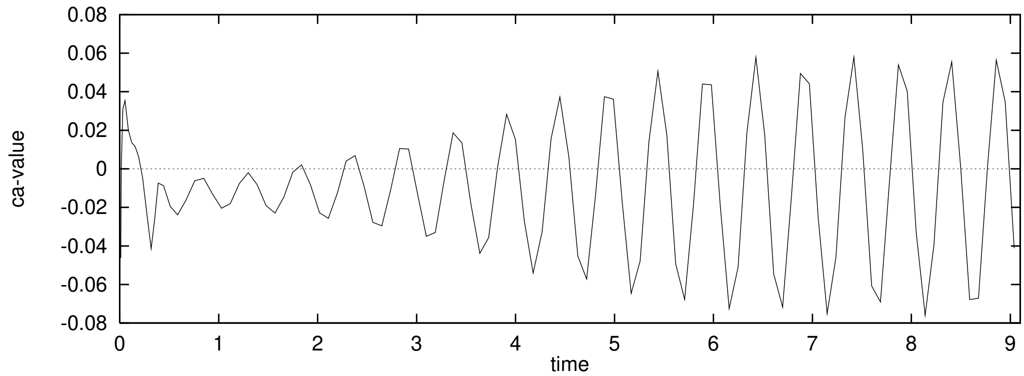

Figure 3.11: Lift values for the nonstationary 3D calculation

We hope, that the presented examples are helpful in understanding the features of FEATFLOW, and give a first feeling how to use it. For more questions, the author is hopefully prepared to your problems.

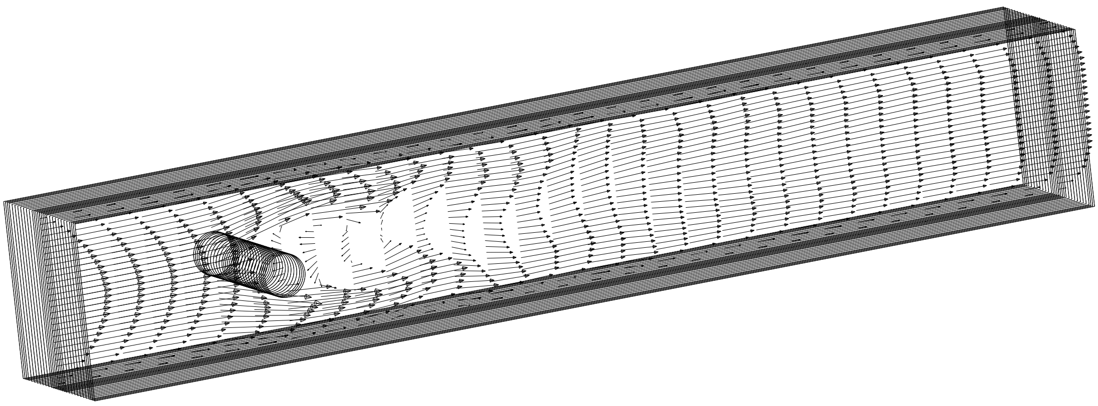

Figure 3.12: Velocity plot in the midplane $z = 0.205$ for $T = 9$

Bibliography

[1] Axelsson, O., Barker, V.A.: *Finite Element Solution of Boundary Value Problems*, Academic Press, 1984

[2] Becker, R., Rannacher, R.: *Finite element discretization of the Stokes and Navier–Stokes equations on anisotropic grids*, Proc. 10th GAMM-Seminar, Kiel, January 14–16, 1994 (G. Wittum, W. Hackbusch, eds.), Vieweg

[3] Blum, H., Harig, J., Müller, S., Turek, S.: **FEAT2D** *. Finite element analysis tools. User Manual. Release 1.3*, Technical report, University Heidelberg, 1992

[4] Chorin, A.J.: *Numerical solution of the Navier–Stokes equations*, Math. Comp., 22, 745–762 (1968)

[5] Crouzeix, M., Raviart, P.A.: *Conforming and non–conforming finite element methods for solving the stationary Stokes equations*, R.A.I.R.O. **R–3**, 77–104 (1973)

[6] Cuvelier, C., Segal, A., Steenhoven, A.: *Finite element methods and Navier Stokes equations*, D. Reidel Publishing Company, Dordrecht 1986

[7] Gresho, P.M.: *On the theory of semi–implicit projection methods for viscous incompressible flow and its implementation via a finite element method that also introduces a nearly consistent mass matrix, Part 1: Theory*, Int. J. Numer. Meth. Fluids, 11, 587–620 (1990). *Part 2: Implementation*, Int. J. Numer. Meth. Fluids, 11, 621–659 (1990)

[8] Girault, V., Raviart, P.A.: *Finite Element Methods for Navier–Stokes Equations*, Springer, Berlin–Heidelberg 1986

[9] Harig, J., Schreiber, P., Turek, S.: **FEAT3D**. *Finite element analysis tools in 3 dimensions. User Manual. Release 1.1*, Technical report, University Heidelberg, 1993

[10] Heywood, J., Rannacher, R., Turek, S.: *Artificial boundaries and flux and pressure conditions for the incompressible Navier–Stokes equations*, to appear in: Int. J. Numer. Meth. Fluids

[11] Kloucek, P., Rys, F.S.: *On the stability of the fractional step–θ–scheme for the Navier–Stokes equations*, SIAM J. Numer. Anal., 31, 1312–1335 (1994)

[12] Müller, S., Prohl, A., Rannacher, R., Turek, S.: *Implicit time–discretization of the nonstationary incompressible Navier–Stokes equations*, Proc. 10th GAMM-Seminar, Kiel, January 14–16, 1994 (G. Wittum, W. Hackbusch, eds.), Vieweg

[13] Rannacher, R.: *Numerical analysis of the Navier–Stokes equations*, Appl. Math., 38, 361–380 (1993)

[14] Rannacher, R.: *On Chorin's projection method for the incompressible Navier–Stokes equations*, in "Navier–Stokes Equations: Theory and Numerical Methods" (R. Rautmann, et al., eds.), Proc. Oberwolfach Conf., August 19–23, 1991, Springer, 1992

[15] Rannacher, R., Turek, S.: *A simple nonconforming quadrilateral Stokes element*, Numer. Meth. Part. Diff. Equ., 8, 97–111 (1992)

[16] Schäfer, M., Turek, S. (with support by F. Durst, E. Krause, R. Rannacher): *Benchmark computations of laminar flow around cylinder*, in E.H. Hirschel (editor), *Flow Simulation with High-Performance Computers II*, Volume 52 of *Notes on Numerical Fluid Mechanics*, 547–566, Vieweg, 1996.

[17] Schieweck, F.: *A parallel multigrid algorithm for solving the Navier–Stokes equation*, Impact Comp. Sci. Engnrg., 5, 345–378 (1993)

[18] Schreiber, P.: *A new finite element solver for the nonstationary incompressible Navier–Stokes equations in three dimensions*, Thesis, University of Heidelberg, 1995

[19] Schreiber, P., Turek, S.: *An efficient finite element solver for the nonstationary incompressible Navier–Stokes equations in two and three dimensions*, Proc. Workshop "Numerical Methods for the Navier–Stokes Equations", Heidelberg, Oct. 25–28, 1993, Vieweg

[20] Thomasset, F.: *Implementation of Finite Element Methods for Navier–Stokes Equations*, Springer, New York 1981

[21] Turek, S.: *A comparative study of time stepping techniques for the incompressible Navier–Stokes equations: From fully implicit nonlinear schemes to semi–implicit projection methods*, Int. J. Numer. Meth. Fluids, 22, 987 – 1011 (1996)

[22] Turek, S.: *On discrete projection methods for the incompressible Navier–Stokes equations: An algorithmical approach*, Comput. Methods Appl. Mech. Engrg., 143, 271 – 288 (1997)

[23] Turek, S.: *Tools for simulating nonstationary incompressible flow via discretely divergence–free finite element models*, Int. J. Numer. Meth. Fluids, 18, 71–105 (1994)

[24] Turek, S.: *Multilevel Pressure Schur Complement techniques for the numerical solution of the incompressible Navier–Stokes equations*, Habilitation Thesis, University of Heidelberg, 1997

[25] Turek, S.: *Multigrid techniques for a divergence–freefinite element discretization*, East-West J. Numer. Math., Vol. 2, No. 3, 229–255 (1994)

[26] Turek, S.: *Efficient solvers for incompressible flow problems: An algorithmic approach in view of computational aspects*, Springer, 1998

[27] Van Kan, J.: *A second–order accurate pressure–correction scheme for viscous incompressible flow*, SIAM J. Sci. Stat. Comp., 7, 870–891 (1986)

A. Appendix: Troubleshooting with FEATFLOW

In the following we list some problems which may apparently occur (as far as the author knows) during the installation and execution of FEATFLOW. The following list cannot contain everything, and the author will be very grateful for showing him even more problems (but, hopefully, with solution strategies).

1) Known problems and errors during installation:

Q.: I have no `gunzip` to decompress my FEATFLOW binary file?
A.: The author may tell you how to get `gzip/gunzip` or can send you the FEATFLOW data in another format!

Q.: I have no FORTRAN77 compiler?
A.: The author may tell you how to get an "older", but "free" FORTRAN77 or the GNU FORTRAN77 compiler!

Q.: I get a `tar` error after decompression?
A.: Your `gzip/gunzip` command may not work well, or you did forget to activate the binary–mode during `ftp` transfer!

Q.: I have problems with the execution of the `make_` shell scripts while installing the system libraries?
A.: Be sure that you start the installation in a `/bin/csh` C–shell!

Q.: The installation stops directly after calling `make_lib`?
A.: Be sure that you the directories `FEATFLOW/object/libraries/libgen` exist!

Q.: I get errors during the compilation process with `make_lib`?
A.: Check your compiler options used, perhaps together with your system administrator!

Q.: My application stops with an error immediately after starting?
A.: Check your `ztime.f` subroutine in `FEATFLOW/source/feat2d/src`!

Q.: If I start the makefiles for CC2D or CC3D in my application, I get the error message that the `VANCA` routines are not linked?
A.: This happens on some computers. Copy the file `FEATFLOW/source/cc2d/src/vanca.f` to `FEATFLOW/source/cc2d/mg`. Then, start there again the shell–script `cc2d_mg.m`. Do the analogous procedure for the 3D case!

2) Known problems and errors during preprocessing:

Q.: I have problems with OMEGA2D?
A.: Check your PC–emulation and that you have (at least) WINDOWS 3.1. If so, ask the author for more advice!

Q.: I have problems with the german manual of OMEGA2D?
A.: Sorry, but in summer 1998 our first version of the DEVISOR will be finished...

Q.: I have problems with TRIGEN2D?
A.: Check in your manual of the actual release, that you marked an admissible set of elements for your desired adaptive refinement strategy!

3) Known problems and errors during solution process:

Q.: I have problems while seemingly the triangulations on every level are created?
A.: The most usual error is that your parametrization or coarse mesh is wrong. For instance, both parametrization curves follow the same direction (error!!!), or the starting point of your parametrization is not captured by a mesh point. Additionally, be sure that all mesh points, which were created while using OMEGA2D, do really belong to an element. In most cases, one of these errors causes your trouble! Another reason might be, if you have more than 1 boundary component, that you must have positioned at least 3 mesh points on every boundary component.

Q.: I have problems with reading an unformatted solution file?
A.: This happens on some computers. Use formatted output!

Q.: The code starts correctly, but before finishing the first iteration step, it stops with an error?
A.: Be sure that the corresponding provided memory size `NNWORK` is large enough!

Q.: The code starts correctly, but the multigrid rates become (almost) identical 1?
A.: Perhaps your mesh is so bad! But in most cases, you prescribed an inflow profile only, and your definition of the boundary part containing natural b.c.'s is wrong (so, your flow cannot get incompressible!!!). Check this in the data file `indat2d.f` (analogously in 3D)!

Q.: My solution schemes (linear muligrid, nonlinear schemes, time stepping schemes) are crashing?
A.: Perhaps your problem is so bad! If you perform a stationary calculation, try the same with the nonstationary code (with adaptively chosen time step size). If you still have problems, then your mesh might be too hard (too large aspect ratios!!!). Check your triangulation. Take the parameter files belonging to the most robust version. However, in most cases it is sufficient to check again the data and the parameter file!

4) Known problems and errors during postprocessing:

Q.: I have none of the proposed graphic tools?
A.: That's a really hard problem! Try to find another one which has similar features, or ask your local computer center, or ask the author ... for the (hopefully soon) freely available AVS EXPRESS modules!

B. Appendix: The FEATFLOW group

We are indebted to the following persons who were involved, with theoretical and practical help or suggestions, to develop FEATFLOW. We hope they won't be too angry about arising questions because of some misleading comments in this manual. However, this is still the first version, and source code and man pages of FEATFLOW will grow hopefully.

Whoever is interested in getting FEATFLOW, or whoever has questions or trouble, please, send an email to S.Turek/Chr.Becker or to

featflow@gaia.iwr.uni-heidelberg.de

List of involved persons:

Christian Becker	University of Heidelberg	cbecker@gaia.iwr.uni-heidelberg.de
Roland Becker	University of Heidelberg	roland@gaia.iwr.uni-heidelberg.de
Heribert Blum	University of Dortmund	blum@math.uni-dortmund.de
Phil Gresho	LLNL Livermore	
Joachim Harig	University of Heidelberg	joachim@gaia.iwr.uni-heidelberg.de
Jaroslav Hron	Charles University of Prague	hron@karlin.mff.cuni.cz
John Heywood	UBC Vancouver	heywood@math.ubc.ca
Susanne Kilian	University of Heidelberg	susanne@gaia.iwr.uni-heidelberg.de
Steffen Müller–Urbaniak	FORD Köln	
Hubertus Oswald	University of Heidelberg	oswald@gaia.iwr.uni-heidelberg.de
Rolf Rannacher	University of Heidelberg	rannacher@gaia.iwr.uni-heidelberg.de
Ludmilla Rivkind	University of Heidelberg	rivkind@gaia.iwr.uni-heidelberg.de
Friedhelm Schieweck	University of Magdeburg	Friedhelm.Schieweck@Mathematik.Uni-Magdeburg.de
Rainer Schmachtel	University of Heidelberg	rainer@gaia.iwr.uni-heidelberg.de
Peter Schreiber	University of Heidelberg	schreib@gaia.iwr.uni-heidelberg.de
Stefan Turek	University of Heidelberg	ture@gaia.iwr.uni-heidelberg.de
John Wallis	University of Heidelberg	wallis@gaia.iwr.uni-heidelberg.de
Owen Walsh	UBC Vancouver	owen@math.ubc.ca

C. Appendix: Future projects in FEATFLOW

In the following we list the current projects which are under development, and which shall be added to one of the next releases of FEATFLOW.

Concerning the improvement of the discretization and solution schemes we are just (or will soon begin with) developing software containing:

- The "mixed coupled" solvers CP2D and CP3D
- Periodic and moving boundaries
- A Galerkin method with adaptive error control in space
- A space–time Galerkin method with adaptive error control
- A parallel version of all codes (for shared and distributed memory machines)
- An improved FEAT version (FEAST !)

Software for a improved preprocessing:

- OMEGA2D as CAD–tool under JAVA (see the DEVISOR !)
- OMEGA3D as CAD–tool under JAVA (see the DEVISOR !)
- TRIGEN2D with mixing of refinement types
- TRIGEN3D with adaptive refinement strategies
- INTPOL2D and INTPOL3D for interpolation between arbitrary meshes

Software for a improved postprocessing:

- "Free" modules of AVS EXPRESS
- Cooperation with other graphic packages

Software for more complex models:

- Adding and testing of Boussinesq- and more complex turbulence models
- Weakly compressible flows
- Non–newtonian and multiphase flows
- Shape optimization (with respect to lift and drag, for instance)

And finally, for the "users":

- Other FORTRAN compilers (GNU, for instance)
- A FORTRAN90 version (see FEAST !)
- A C or C++ version (?)
- Industrial and commercial applications !!!

If you have more suggestions, please, let them be known the authors.