Efficient Multigrid Poisson Solvers in General Coordinates on Tensorproduct Meshes

(Effiziente Mehrgitterverfahren für Poisson-Probleme in allgemeinen Koordinaten auf Tensorproduktgittern)

Diplomarbeit

von

John Wallis

Betreuer:

PD Dr. Stefan Turek

Fakultät für Mathematik

Universität Heidelberg

Januar 1999

Contents

Preface

The aim of this work is the numerical and computational analysis of (iterative) solvers for Poisson problems. Particular emphasis is made on the following aspects:

1. Different coordinate systems:

Problems are often formulated in coordinate systems which are compatible with the problem. For example, for star simulations in astrophysics, cylindrical or spherical coordinates are preferred because they are closer to the geometry. Choosing such special coordinate systems allows the transformation to highly structured tensorproduct meshes which result in higher computational efficiency. On the other hand, this transformation into such coordinate systems leads to more complex coefficients in the Partial Differential Equations (PDE), such that the corresponding solvers must be tuned appropriately due to the resulting anisotropies.

2. Different geometrical complexity:

Real life computations can be geometrically complex if the shape of the domain has to be adequately included. Additionally, small scale structures or boundary layers have to be approximated sufficiently accurate such that the resulting meshes can be very anisotropic. Again, the different solvers have to be checked with respect to their robustness behavior.

3. Different computational complexity:

Realistic computational simulations, such as in Computational Fluid Dynamics (CFD), can lead to very large discrete systems of equations due to geometrical but also accuracy reasons. In light of this, a corresponding solver

must be extremely fast to handle up to millions of unknowns and thousands of (implicit) time steps.

Concluding all these problems, the task of this work is to develop very efficient solvers for Poisson-like problems which can handle

- complex meshes with large anisotropies and small-scale structures
- various coordinate systems with anisotropic coefficients

while providing at the same time a significant percentage of the high computing power on modern hardware platforms to solve the resulting huge linear systems.

To be more precise: As part of the FEAST project which is currently under development at our institute, this work develops and studies various solvers on generalized tensorproduct meshes, here for conforming bilinear finite element (FEM) discretization. Since the computational domain is broken up into such smaller quadrilateral patches, the so-called 'macros' for our solution process, these 'tensorproduct' solvers are one of the key tools for the performance of the numerical simulation.

Some major application fields for the FEAST software and therefore for us are problems arising in Computational Fluid Dynamics ('incompressible Navier Stokes equations') and in hydrodynamics in astrophysics. These different physical situations have all common that the complete problem, such as for pressure, velocity or density, can be reduced among others to Poisson-like subproblems ('Pressure-Poisson', 'gravity problem'). Thus, for solving large problems of this type the derived multigrid methods are an important tool.

The paper is organized as follows: The first chapter introduces the aims and concepts of the FEAST project. Based on the described FEAST philosophy, we introduce different types of tensorproduct meshes and give a plausible classification with respect to anisotropies. These discretization aspects are followed by some physical problems which are the background for our mathematical approaches: different coordinate systems are introduced and the corresponding formulations are made for each of them.

In the second chapter, the components of the linear solvers and the underlying FEM discretization are described in more detail. Comparison between

Gaussian elimination (single grid), Krylov-space methods and multigrid algorithms is made. Hereby, we concentrate on multigrid schemes as our preferred solvers. Finally, implementation issues, such as matrix storage and machine-optimized Numerical Linear Algebra components for such special matrices, are examined since they are mainly responsible for the high performance rates of our approaches.

The third chapter contains the results of the extensive tests using the multigrid solvers. It will be shown that for low anisotropies all smoothers, including Jacobi and Gauß-Seidel preconditioning, do well for cartesian and cylindrical coordinates. However, for larger mesh anisotropies, respectively, for spherical coordinates, more sophisticated smoothers of tridiagonal type or ILU have to be taken. As a conclusion we can state, that our linewise Gauß-Seidel variants show the best results if we take into account the numerical and computational run-time behavior, and this for all coordinate systems.

After the final conclusions from our numerical studies, we shortly print some of the developed software which is the basis of all our computations.

Chapter 1

Motivation and problem description

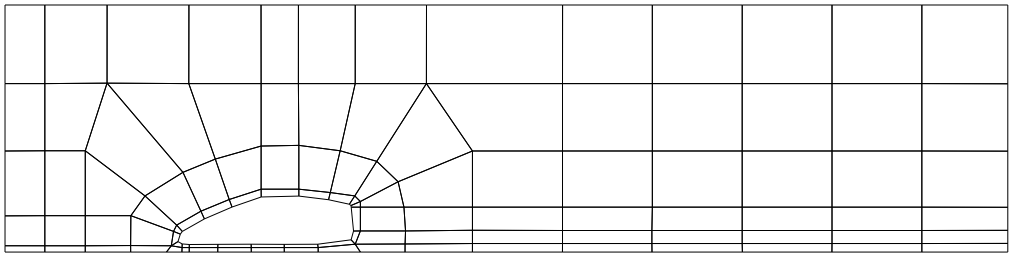
A current trend in the numerical simulation of PDE's, here for FEM approaches, is the use of object-oriented techniques and adaptive methods in any sense. Unfortunately, for the structures involved for handling grid and matrices, old-fashioned methods are still being used which ignore hardware capabilities of modern hardware platforms. Thus peak performance rates, today around 1 GFLOP/s, are seldom reached, and the gap to 'real life' approaches is increasing when one takes current hardware developments into consideration.

High performance calculations often require exploitation of caching and pipelining techniques with sequential data. The implementation of theses techniques is often clear for Finite Difference techniques, but for more complex Finite Element approaches, this is not the case. This difference often leads to unreasonable computation times for 'real life' problems such as CFD in 3D. To help diminish this problem is one of the primary aims of the FEAST project ('Finite Element Analysis and Solution Tools') (see Section 1.1).

An example for performance loss is shown in the following: one of the main components in numerical computations is Matrix-Vector (MV) operations. They take up 60-90 percent of CPU time or more in iterative solvers. To approach these operations, sparse methods are employed (see [10]). These

make the use of index arrays and a long array holding the 'nonzero' elements of the matrix. This universal sparse technique can be applied to very general meshes with arbitrary numbering. Consequently, this storage technique can hardly gain from the benefits of caching and pipelining and thus more time is lost accessing memory.

In the paper [3] are computational studies for different matrix storage techniques. Using our CFD code FEATFLOW, tests examine the following 'Flow around a car' problem.



The table below shows results for the 'two level' (TL) numbering, the Cuthill McKee (CM) algorithm which minimizes bandwidth and an arbitrary 'stochastic' numbering of the unknowns. All have in common the standard sparse 'Compact Storage Rowwise' CSR technique. Cost for arithmetic operations and memory access are identical. NEQ denotes the number of unknowns.

Computer	NEQ	TL	CM	stochastic
	13,688	20	22	19
SUN E450	54,256	15	17	13
($\sim$ 250 MFLOP/s)	216,032	14	16	6
(CSR)	862,144	15	16	4

The results of this table show that:

1. Different numbering strategies can lead to identical numerical results, but have huge differences in CPU time.

2. Sparse MV techniques are slow and depend heavily upon problem size and the memory access techniques employed.

3. Multigrid runtime behavior is not that of the expected mathematical theory which predicts a linear relationship.

For scalar PDE's on (logical) tensorproduct meshes in 3D with trilinear FEM spaces (matrix bandwidth=27), the following 'strange' MFLOP/s rates are received.

Computer	NEQ	TL	CM	stochastic
IBM RS6000/597	17^3	86	81	81
(160 Mhz)	33^3	81	55	16
'P2SC'	65^3	81	14	8
IBM RS6000/590	17^3	42	42	42
(66 Mhz)	33^3	41	39	27
'POWER2	65^3	41	17	7
DEC/21164	17^3	54	31	29
(433 Mhz)	33^3	51	16	10
'CRAY T3E'	65^3	49	13	8
PENTIUM II	17^3	30	28	28
(400 Mhz)	33^3	30	26	24
'ALDI PC'	65^3	30	23	19

- MFLOP/s rates are far away from 'peak performance'.
- Size and type of memory access is significant for the MFLOP/s rates.
- Even when the problem fits entirely into cache, index access leads to losses.
- Old processors (590) can be faster than new ones (597).
- Supermarket PC's can run faster than processors in supercomputers.

In contrast, the highly structured MV techniques of FEAST, due to exploited vectorization facilities and data locality, possess MFLOP/s rates up to 50 times better! The next table compares the use of variable matrix entries and constant band entries on tensorproduct meshes which can be additionally exploited by this new approach.

Computer	NEQ	'var'	'const'
IBM RS6000/597	17^3	188	480
(160 Mhz)	33^3	172	393
'P2SC'	65^3	176	390
IBM RS6000/590	17^3	102	195
(66 Mhz)	33^3	94	175
'POWER2	65^3	94	176
DEC/21164	17^3	103	404
(433 Mhz)	33^3	101	313
'CRAY T3E'	65^3	101	268
PENTIUM II	17^3	51	180
(400 Mhz)	33^3	51	137
'ALDI PC'	65^3	48	124

The next table, from the '1997 Semiconductor Roadmap', see [11], shows the expected development of the actual processor technology. It demonstrates that 'machine-oriented' algorithmic and implementation techniques will be absolutely necessary: the key techniques for the near future will be to better exploit data locality, internal parallelism and vectorization.

1997 National Technology Roadmap for Semiconductors							
Year of 1st Shipment	**1997**	1999	2001	2003	2006	2009	**2012**
Local clock (Mhz)	**750**	1250	1500	2100	3500	6000	**10K**
Chip size (mm^2)	**300**	340	385	430	520	620	**750**
Feature size (nm)	**250**	180	150	130	100	70	**50**
Number of chip I/O	**1450**	2000	2400	3000	4000	5400	**7300**
Transistor/chip	**11M**	21M	40M	76M	200M	520M	**1.4B**

From this table, single processors will be 10-15 times faster and will possess up to 100 times more transistors than today. This is the equivalent of packing 1000 modern PC's into one chip and will perform better than a complete CRAY T3E today and may reach TFLOP/s rates. Therefore, our aim in the FEAST project is the development of special numerical and implementation techniques to exploit this high computing power.

1.1 The FEAST project

This work is part of the FEAST project. Its primary aim is the combination of highly tuned Linear Algebra tools and FEM simulation strategies such as:

- consequent application of (recursive) 'Divide and Conquer' strategies
- solution strategy SCARC as 'new' generalized multigrid (MG) and domain decomposition (DD) technique
- hierarchical data and solver structures, but also hierarchical (!) matrix structures
- integration of low level machine-optimized Linear Algebra routines for such sparse banded matrices

and its main goal is easy implementation of many 'real life' problems with special attention given to

- achieving (almost) peak performance on modern processors
- typical multigrid behavior (with respect to efficiency and robustness)
- 'low-level' parallelization directly included
- ease of modification to include concepts as adaptivity and a posteriori error control

FEAST is designed for high-performance applications with industrial background with special attention given to *Computational Fluid Dynamics* (CFD). As a result, efficiency and robustness are at the forefront of our approaches. FORTRAN 77/90 is used which allows the adoption of the predecessor packages FEAT2D, FEAT3D and FEATFLOW. Further information on FEAST may found in [1], [4], [13] and [14]. See also the FEATFLOW homepage at

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

One of the most important aspects in FEAST is the use of 'Divide and Conquer' strategies. A 'global' problem is subdivided into smaller independent problems on separate 'patches'. Hereby, structured parts may be locally exploited and anisotropies are hidden. On these anisotropic 'patches', sparse techniques are employed. On the structured 'patches', higher performance is exploited. As a result, the goal is to minimize the number of sparse areas and to apply, where possible, all Numerical Linear Algebra routines to the structured 'patches'. The simulation tool is then realized in three major steps:

1. The design of the skeleton for the recursive splitting into local/global levels
2. The implementation of the typical FEM facilities on the 'low level' patches
3. The development of 'reference element solvers' on the highly structured 'low level' patches

The respective data, solver and matrix structures may be found in [1], [4], [13] and [14]. The aim of this paper is to help realize the second and third parts: The implementation of typical FEM facilities on the 'low level' patches and particularly the development of 'reference element solvers' on the highly structured 'low level' patches. In our context, these are convex quadrilaterals (='macros') with logically equivalent tensorproduct meshes which allow local mesh adaption if necessary.

A principle strategy of FEAST is to do as much as possible on such (logically equivalent) tensorproduct meshes (m*m) for general quadrilaterals. This work studies several multigrid solvers in generalized coordinates for these geometries. We use conforming finite elements that have their degrees of freedom defined in the vertices. For individual elements, a 2x2 Gaußian formula for integration is used to compile a 'nine-star' matrix. Also available for the matrix compilation is the method that employs the trapezoid rule. This can result in the typical 'five-star' matrix as often found in Finite Difference Methods.

For the program to be feasible to use for realistic needs three basic requirements are needed for the multigrid solver:

1. The basic fixed point iteration $x^{l+1} = x^l - C^{-1}(Ax^l - b)$ the preconditioned Richardson iteration. This procedure is at the heart of the smoothing and takes up the largest part of CPU time. It must be optimally implemented not only for domain shape, aspect ratio, etc, but also for the platform on which the program is run, to optimize vector operations with cache considerations. The numerical convergence behavior of this iteration is directly related to the iteration matrix

$$M = I - \omega C^{-1} A$$

   and hence to the more or less clever choice of C.

2. Problems to be solved are massive (up to about 100,000 unknowns and more), thus high CPU-cost arise so optimizations must be employed where possible.

3. The program should be robust and stable in order to challenge grids with extreme aspect-ratios and varying physical situations.

What we hope to gain is:

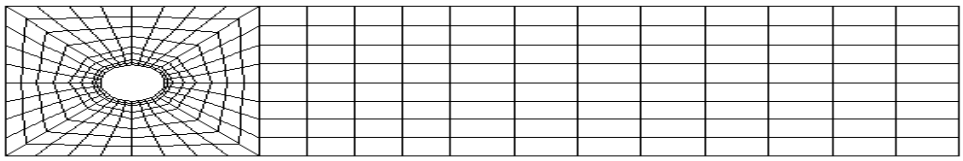
- less storage amount: since vertices are numbered row-(column-)wise fewer index tables are required and potentially even the storage of complete matrices can be saved.

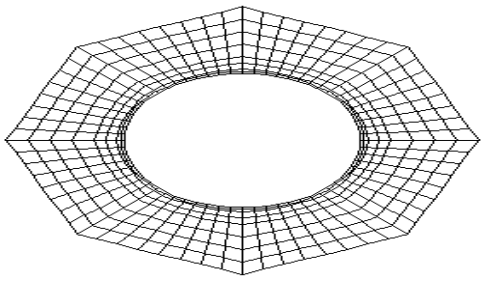
- 'reference element' solvers: the complete problem is divided into smaller problems on our 'patches' that can be individually optimized and controlled via a priori studies.

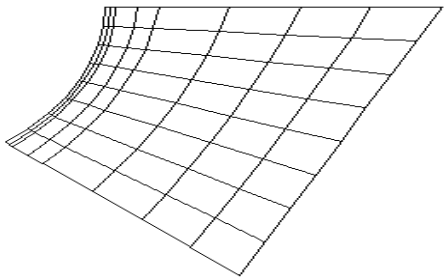
To ensure robustness, all parts of this work are based upon tested and optimized routines of FEAST. Each subprogram has been optimized for optimal efficiency and performance: matrix-vector, vector-vector, prolongation, restriction, cache, etc. Therefore, the aim is multigrid on one quadrilateral, which is triangulated via a generalized tensorproduct mesh!

1.2 Generalized tensorproduct meshes

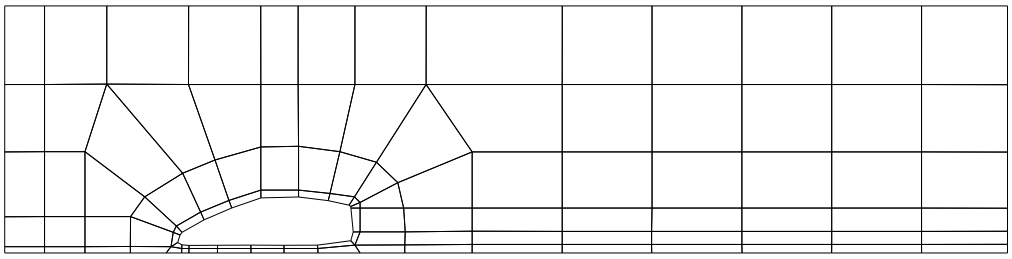
A typical example used in CFD simulations is a 2d channel, shown on the next page, with an obstacle for a fluid moving towards the right. One can expect the most 'turbulence' near the obstacle, thus a higher refinement in this region is preferable. Each small 'patch' could be solved for independently. The first figure shows the initial 'macro' mesh with 22 'macros'. The next figure shows the mesh after 2 global refinements. Below is shown the area closest to the circle magnified and besides it one 'macro', three times refined with anisotropic refinement towards the circle.





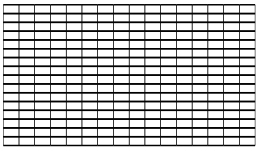


In the figure below is a rough discretization of the 'Flow around a car' problem with 106 'macros'.

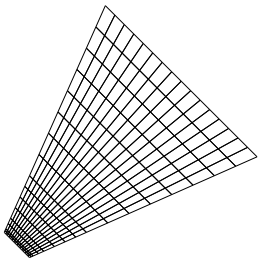


Below are a few typical 'macros' from the problem. Elements far away from the car do not necessarily need a special refinement and can be close to isotropically refined. On the other hand, elements nearer to the car and boundaries may need more adaptation due to boundary effects. Here, the refinement is made in direction of the car and of boundaries which lead to higher aspect ratios for these 'macros'. For given aspect ratios an appropriate solver has to be chosen to optimize speed and/or memory requirements. On each 'macro' different refinements and levels may be used to optimize approximation and memory use.

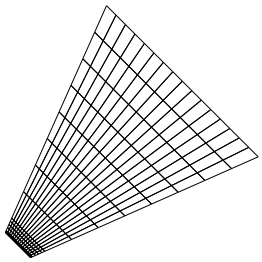
elements far behind

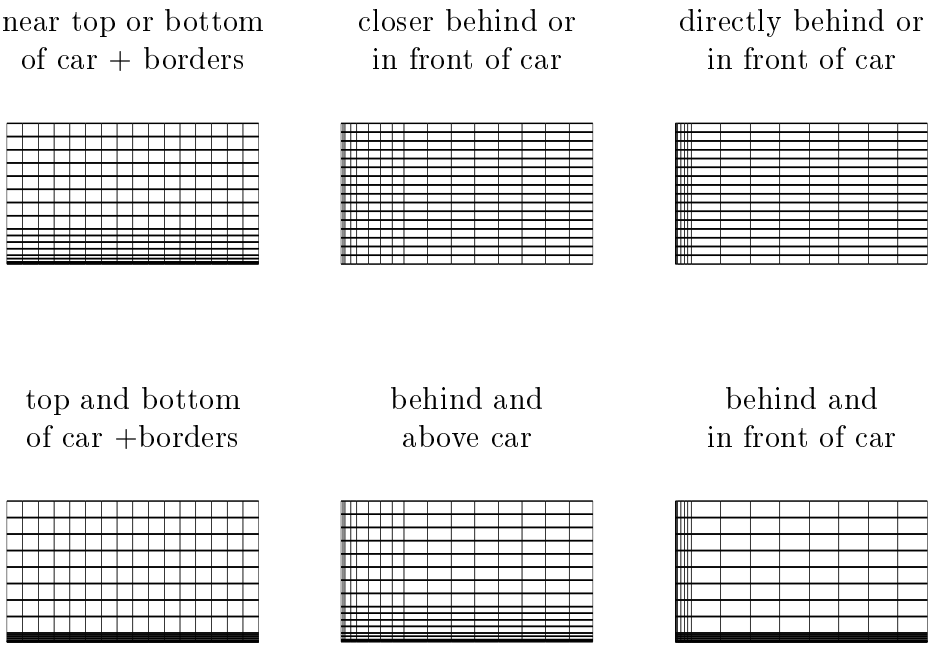


near corners of car

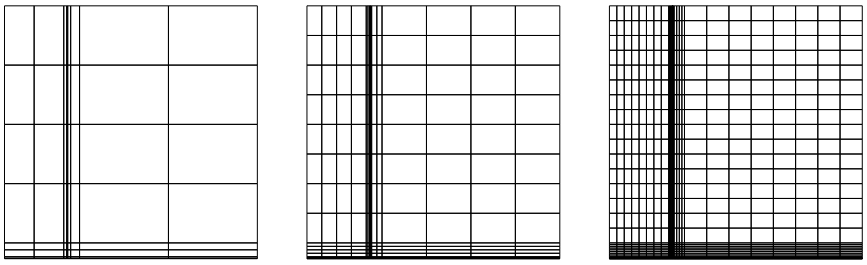


closer to corners





The next figures show special refinements of a square domain on different levels. Such meshes are used for calculations for accretion disks in astronomy. However, we used here different scaling for better showing of the mesh, and we did not adapt precisely to the physics. The star is located on the left side of the mesh where the x-axis represents the radius from the star and the y-axis extends from the equator plane. Refinement is made a distance away from the star to better approximate effects stemming from shock phenomena. The computations would be made using cylindrical or spherical coordinates, but on generalized tensorproduct meshes.



The local refinement of a quadrilateral (='macro') is described via a routine defined by the following algorithm which is shown here for that case that a line with endpoints $[z_0, z_1]$ is to be refined L times. The generalization to arbitrary quadrilaterals is straightforward (see [9]).

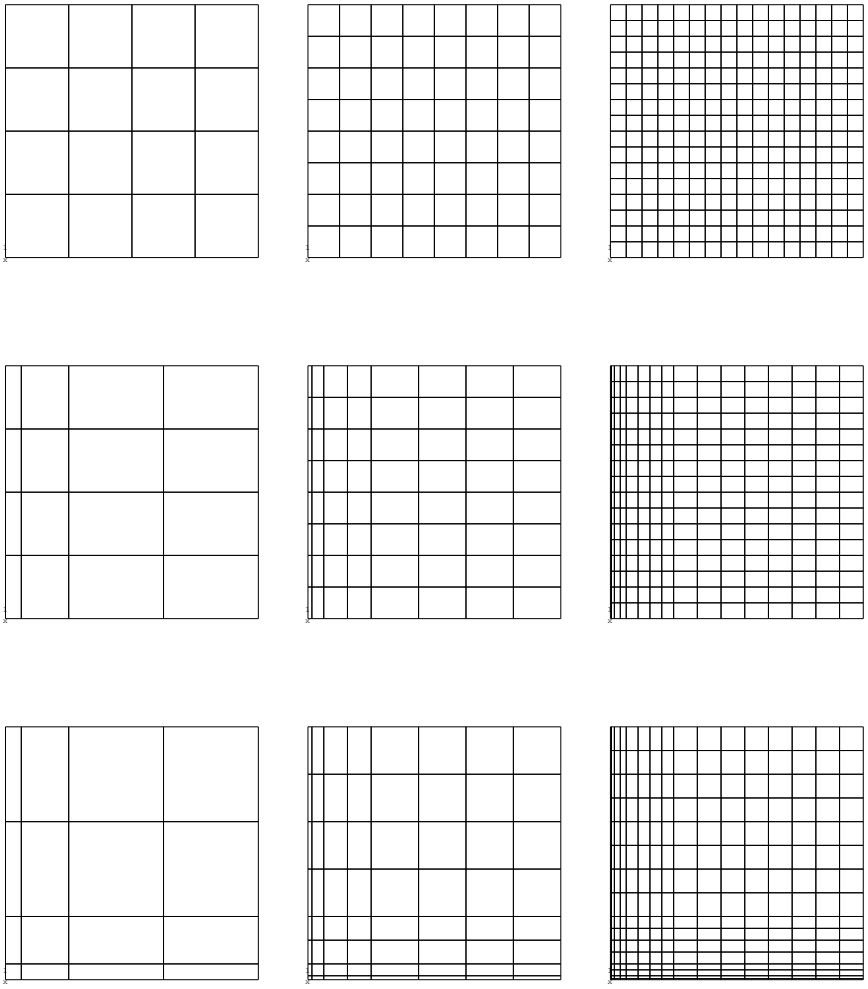
Procedure $grid - refine^L(\nu_1, \nu_2, \nu_3)$

For l=0,...,L

1. l=0: Initialization
   - set $x_0^0 := z_0$
2. l=1: First refinement
   - $x_0^1 := z_0$
   - $x_1^1 := \nu_1(x_0^0 + x_1^0)$
   - $x_2^1 := x_1^1$
3. l>1: Main steps
   - $x_0^l := z_0$
   - $x_1^l := \nu_2 \frac{x_0^{l-1} + x_1^{l-1}}{2}$
   - $x_{2i}^l := x_i^{l-1}; \qquad i = 1, ..., 2^{l-1}$
   - $x_{2i+1}^l := \frac{x_i^{l-1} + x_{i+1}^{l-1}}{2}; \qquad i = 1, ..., 2^{l-1} - 1$
4. l=L: Last refinement
   - $x_0^l := z_0$
   - $x_1^l := \nu_3 \frac{x_0^{l-1} + x_1^{l-1}}{2}$
   - $x_{2i}^l := x_i^{l-1}; \qquad i = 1, ..., 2^{l-1}$
   - $x_{2i+1}^l := \frac{x_i^{l-1} + x_{i+1}^{l-1}}{2}; \qquad i = 1, ..., 2^{l-1} - 1$

With this routine one has a maximum aspect ratio of $0.5/\nu_1\nu_2^{l-1}\nu_3$ for refinement in one direction and $(1-\mu_1\mu_2)/\nu_1\nu_2^{l-1}\nu_3$ for refinements done in both directions, where the ν_* are refinement parameters equal or finer to the μ_* parameters. For the condition that $\frac{\nu_1}{2} < \nu_2 - \frac{\nu_1}{2} < \nu_3 - \nu_2 < 1 - \nu_3$, the smallest interval is $h_{min} = \frac{\nu_1\nu_2^{l-1}\nu_3|z_0-z_1|}{2^{l-1}}$ and the greatest interval is $h_{max} = \frac{(1-\nu_1\nu_2)|z_0-z_1|}{2^{l-1}}$.

A few examples of the refined elements (up to 4 refinements) used for the multigrid method can be seen below.



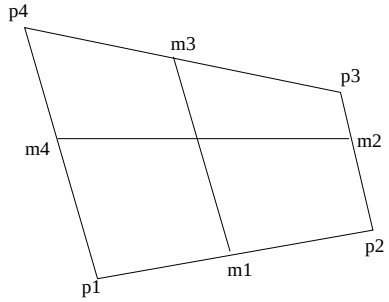
Figures show (0.5, 1.0, 1.0)-, (0.5, 0.5, 0.5)- and bi(0.5, 0.5, 0.5)-refinement of a square domain on levels 2,3,4.

The quadrilaterals have to be convex, and the coordinates of each of the corners may be freely defined. The refinement within the element is controlled via the three parameters from above.

The grid is numbered row-wise and a function that 'transposes' the grid is available for row- or columnwise numbering for tridiagonal/ADI preconditioned solvers. Since the grid has row-wise numbering, it lends itself well on implementing band-structure matrix techniques, thus the use of the optimized band routines from FEAST can be employed. Also, the matrix allows for realizing block-solve technique implementations as shown in the previous examples.

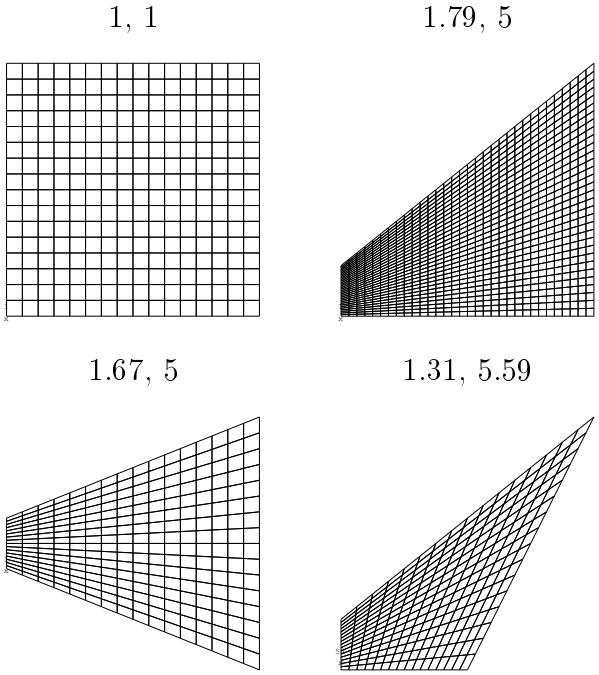
Aspect ratios (AR) play an important role in the behavior of the implemented solvers. The classical aspect-ratio used is found by making a fraction of the 2 lengths found when measuring the distance of the two opposing midpoints of an element. The maximum over all elements is taken and if it is less than 1, it is inversed so that the AR is always greater than 1. Another definition of aspect ratio can be made be taking the quotient of the lengths of two opposing sides.

The figures below show the classical and new aspect ratios for the same respective element and demonstrate why we had to modify the definition of aspect ratio. Further examples can be found in section 3.2.



$$AR_{classical}=max(\frac{|m3-m1|}{|m4-m2|}, \frac{|m4-m2|}{|m3-m1|})$$

$$AR_{new}=max(\frac{|p2-p1|}{|p4-p3|}, \frac{|p4-p3|}{|p2-p1|}, \frac{|p4-p1|}{|p3-p2|}, \frac{|p3-p2|}{|p4-p1|}, AR_{classical})$$



The stiffness matrix is directly defined by the weights/distances of each grid point. For elongated elements, factors in one direction may become very small. For example, for row-wise numbering, for elements narrow with respect to the x-coordinates and wide with respect to the y-coordinates, the corresponding matrix will converge to a tridiagonal matrix with coefficients corresponding to the x-direction only.

Since a domain can have many regions with more and less activity, one expects most activity behind objects or along walls, thus different logical refinement strategies are employed. Then, the complete domain is subdivided into smaller macros and each of these mini-domains is solved for separately and reassembled then to solve the global problem (see [9]). However, the robust and efficient process on each of these macros, respectively 'mini-domain', is our task in the following.

1.3 Examples for physical configurations

1.3.1 Incompressible Navier-Stokes equation

The Navier Stokes equation has a real-life character and all the mathematical complexities and difficulties which makes it attractive for further numerical study. The incompressible Navier Stokes equation (plus boundary/initial conditions) is written as

$$u_t - \nu\Delta u + u\nabla u + \nabla p = f,\ \nabla \cdot u = 0$$

with velocity u and pressure p, so u_t represents the time derivative of u, Δ the Laplace operator and $\nabla\cdot$ the divergence and ∇ the gradient operators, respectively. A typical approach to solving such problems is to employ projection-like methods (see [12]).

After performing space and time discretization, the corresponding problem in each time step can be written in matrix form $\binom{S\ B}{B^T 0}\binom{u}{p} = \binom{f}{0}$ where S is the matrix holding the coefficients for u, B equals the gradient (∇)matrix and B^T is the divergence matrix. A typical strategy is the following: An intermediate value $\tilde{u}$ for u at a new time level is found by solving

$$S\tilde{u} = \tilde{f} - kBp_{old},$$

where k denotes the time step. Here, S represents the momentum equation with or without linearization strategies. Then we update the pressure ('Pressure-Poisson problem') via:

$$-\Delta_h q = \frac{1}{k} B^T \tilde{u}$$

Finally, the new pressure and the new 'divergence-free' velocity vectors are updated. The parameter σ is in the range (0,2], typically $\sigma = 1$.

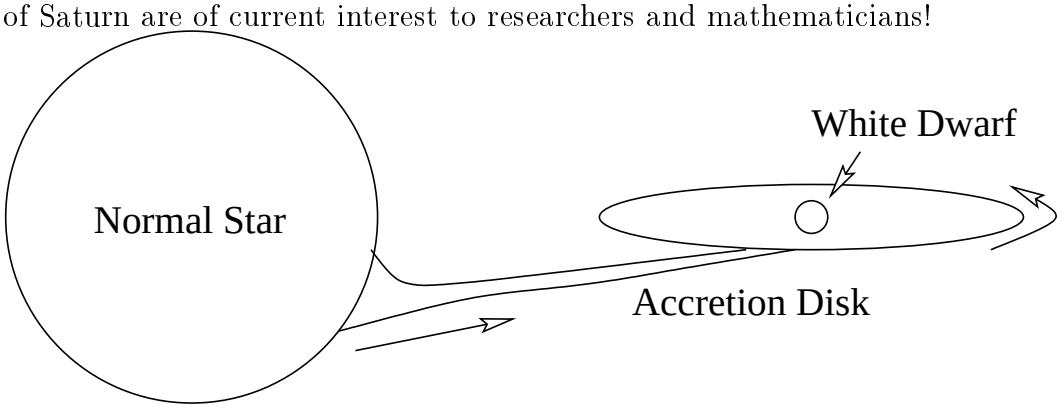
$$p_{new} = p_{old} + \sigma q, \quad u_{new} = \tilde{u} - kBq$$

and we can define the values on the new time level as p_{new} and u_{new}.

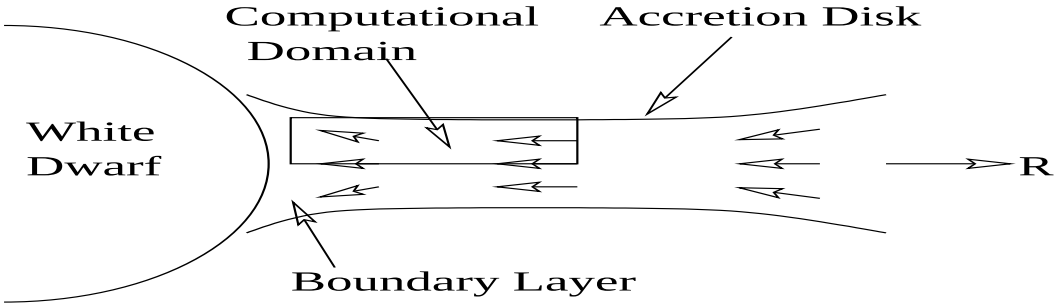
As typical benchmarks show (see [12]), the major task in many codes is to solve this Pressure-Poisson problem, which is typically situated on huge and complex meshes, and the process in fully nonsteady simulations has to be repeated very often.

1.3.2 Astrohydrodynamics

In astrohydrodynamics, simulations of stars, accretion disks, or even the rings of Saturn are of current interest to researchers and mathematicians!



Accretion disks form when matter ascends upon a large central mass. Because of the rotational momentum of the matter, for example in a rotating binary star system, the matter does not fall directly onto the central mass, but loops around following Newton's laws and forms a ring. The ring eventually flattens because of its self-gravity, similar to the rings of Saturn.



The basic equations in astrohydrodynamics are the continuity equation

$$\frac{d\rho}{dt} + \frac{\partial}{\partial x}(\varrho v) = 0,$$

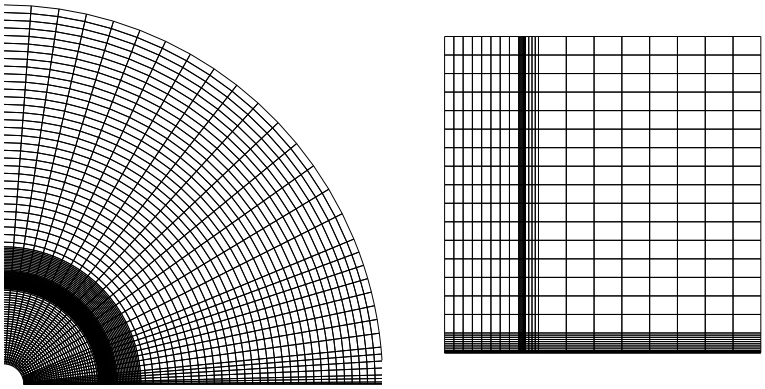
the Euler equation

$$\frac{\partial v}{\partial t} + v\frac{\partial v}{\partial x} + \frac{1}{\varrho}\frac{dp}{dx} + \nabla\Phi = 0$$

and for the gravity potential Φ, the Poisson equation

$$\Delta\Phi = -4\pi G\varrho.$$

Again, this Poisson problem is the major aim of this work! For accretion disks, higher refinement is preferred along the equator and near the central star. Sometimes refinement is made a distance away from the star because a shock may form here. The following picture shows a prototypical mesh used for the simulation (in a 2d projection) and a cartesian representation after transformation.



1.4 Description of the mathematical problems

1.4.1 Boundary conditions

A possibility for boundary conditions in many CFD configurations is to define the inflow side via Dirichlet (speed) conditions and the outflow elements with natural Neumann boundaries, or the boundary parts could similarly be defined via pressure-drop conditions (see [8]). The top and bottom edges could be defined to Dirichlet conditions to simulate a wind-tunnel. Elements far behind the car can be approximated using fewer grid points and since one can expect the most activity near and behind the car, elements near the car should be refined higher and in direction of the car.

There are three types of boundary conditions on the macros: Dirichlet, Neumann and 'mirror'. Dirichlet conditions define a value at a border point, these can be the speed of an incoming material or a pressure at a border, Neumann conditions defined the change of a variable at the border. For free

flowing fluids, this is often similar to $\nu\frac{\partial u}{\partial n} + p \cdot n = 0$. 'Mirror boundaries' are a 'combination' of Dirichlet and Neumann conditions (see [9]).

Boundary conditions for an accretion disk might be defined as follows. At the outer rim of the disk a small amount of material must continuously 'feed' the accretion disk, thus Dirichlet boundary is employed here. At the equator, 'mirror' boundary conditions could be used (assuming top/bottom symmetry). At the central axis, 'mirror' boundarys are employed. This work does calculations on 2D domains, thus for spherical and cylindrical coordinates, symmetry is assumed around the z axis (see [6]).

1.4.2 Stopping Criteria

The iteration scheme stops if the defect D=Au-f in the Euclidean norm $||D|| \leq$ TOL. Let $\varphi_h^i \in V_h$ be the basis functions for denoting the FEM space, then components of the defect vector fulfill:

$$D_h^{(i)} = a_h(u_h, \varphi_h^{(i)}) - (f, \varphi_h^{(i)}) = (L_h u_h - f, \varphi_h^{(i)}).$$

Now, we define the finite element residual $res_h \in V_h$ via L^2-projection:

$$(res_h, \varphi_h) = (L_h u_h - f, \varphi_h) \ \forall \varphi_h \in V_h.$$

The relation between the defect vector D_h and finite residual res_h is

$$D_h = M_h res_h$$

so we have

$$||res_h||^2 \sim \sum_i (M_h^{(i)})^{-1} (D_h)^2,$$

$$||D_h^{(i)}||_E^2 = \sum_i (D_h)^2$$

M_h is thereby the mass matrix created by 'lumping' which is equivalent to using the trapezoid rule. If the grids are isotropic both of these expressions are of the same order, up to a scaling factor of order of h^2, but varies with increasing (local) aspect-ratios. The first of the expressions is preferred because the topologie (weighting factors) of the grid is considered, thus it represents

a better stopping criteria for the iteration relative to the accuracy wanted and is independent of the number of unknowns (NEQ).

The difference between the two norms is noticed in the size of the defect and the start defect-values. For the tables below, D is the L_2 norm of the defect while M represents the mass-matrices produced by

$M = \int \varphi_i \varphi_j \cdot dxdy$ and

$M_r = \int \varphi_i \varphi_j \cdot \rho d\rho dz$ for cylindrical coordinates, and

$M_r = \int \varphi_i \varphi_j \cdot \rho^2 sin(\vartheta) d\rho d\vartheta$ for spherical coordinates.

As smoother in multigrid program, we take GSADI (see Section 3.1.5), and we perform the strategy of "gain 8 digits". The tables show the starting residual and the convergence rates, w.r.t. the different norms involved.

GSADI, Isotropic (0.5, 1.0, 1.0), Dirichlet, smooth=2									
Cart.	D			M			M_r		
Level	5	6	7	5	6	7	5	6	7
iter 0	.35+0	.17+0	.87-1	.11+2	.11+2	.11+2	same as M		
conv.	.17-1	.17-1	.17-1	.17-1	.17-1	.17-1			
GSADI, Isotropic (0.5, 1.0, 1.0), Dirichlet, smooth=2									
Cyl.	D			M			M_r		
Level	5	6	7	5	6	7	5	6	7
iter 0	.19+0	.96-1	.48-1	.27+2	.12+2	.85+1	.62+1	.62+1	.62+1
conv.	.92-1	.12+0	.14+0	.93-1	.13+0	.17+0	.92-1	.12+0	.14+0
GSADI, Isotropic (0.5, 1.0, 1.0), Dirichlet, smooth=2									
Spher.	D			M			M_r		
Level	5	6	7	5	6	7	5	6	7
iter 0	.97-1	.49-1	.25-1	.21+2	.10+2	.80+1	.31+1	.31+1	.31+1
conv.	.60-1	.80-1	.10+0	.59-1	.89-1	.11+0	.60-1	.80-1	.10+0

GSADI, Anisotropic (0.5, 0.5, 1.0), Dirichlet, smooth=2									
Cart.	D			M			M_r		
Level	5	6	7	5	6	7	5	6	7
iter 0	.41+0	.20+0	.10+0	.11+2	.11+2	.11+2	same as M		
conv.	.25-1	.28-1	.29-1	.26-1	.35-1	.38-1	same as M		
GSADI, Anisotropic (0.5, 0.5, 1.0), Dirichlet, smooth=2									
Cyl.	D			M			M_r		
Level	5	6	7	5	6	7	5	6	7
iter 0	.23+0	.18+0	.59+0	.79+1	.79+1	.80+1	.62+1	.62+1	.62+1
conv.	.61-1	.66-1	.68-1	.92-1	.11+0	.12+0	.69-1	.28-1	.27-1
GSADI, Anisotropic (0.5, 0.5, 1.0), Dirichlet, smooth=2									
Spher.	D			M			M_r		
Level	5	6	7	5	6	7	5	6	7
iter 0	.12+0	.60-1	.30-1	.16+2	.90+1	.78+1	.31+1	.31+1	.31+1
conv.	.46-1	.63-1	.80-1	.54-1	.82-1	.96-1	.47-1	.64-1	.80-1

GSADI, Anisotropic (0.25, 0.25, 0.5), Dirichlet, smooth=2									
Cart.	D			M			M_r		
Level	5	6	7	5	6	7	5	6	7
iter 0	.47+0	.24+0	.12+0	.11+2	.11+2	.11+2	same as M		
conv.	.35-1	.32-1	.31-1	.46-1	.42-1	.48-1	same as M		
GSADI, Anisotropic (0.25, 0.25, 0.5), Dirichlet, smooth=2									
Cyl.	D			M			M_r		
Level	5	6	7	5	6	7	5	6	7
iter 0	.26+0	.13+0	.66-1	.79+1	.79+1	.79+1	.62+1	.62+1	.62+1
conv.	.46-1	.48-1	.46-1	.89-1	.11+0	-	.55-1	.63-1	.67-1
GSADI, Anisotropic (0.25, 0.25, 0.5), Dirichlet, smooth=2									
Spher.	D			M			M_r		
Level	5	6	7	5	6	7	5	6	7
iter 0	.11+0	.67-1	.34-1	.16+2	.90+1	.78+1	.31+1	.31+1	.31+1
conv.	.41-1	.58-1	.70-1	.62-1	.12+0	.10+0	.44-1	.64-1	.74-1

The classical defect norm obviously does not incorporate the mesh geometry and mesh size, such that with increasing the mesh level the initial defects automatically get smaller. Concluding these results, we clearly prefer the residual-load approach, involving the corresponding mass matrix for the following calculations.

1.4.3 Coordinate systems

A coordinate system is defined by a basis. Any position in the coordinate space is defined by a unique parameter representation.

The unit of movement e_{x_i} for any direction x_i is given by $\frac{\partial \overrightarrow{r}/\partial x_i}{|\partial \overrightarrow{r}/\partial x_i|}$ or $\partial \overrightarrow{r}/\partial x_i = h_i \overrightarrow{e}_{x_i}$, where $h_i = |\partial \overrightarrow{r}/\partial x_i|$, i=1..3.

Let $\nabla = (\overrightarrow{e}_1 \frac{\partial}{\partial x_1} + \overrightarrow{e}_2 \frac{\partial}{\partial x_2} + \overrightarrow{e}_3 \frac{\partial}{\partial x_3})$ be an operator over a scalar field in a coordinate system. The result of $\overrightarrow{\nabla}\phi$ is a vector and is called grad (gradient), and $\overrightarrow{\nabla} \cdot \overrightarrow{r}$ is a scalar and is called div (divergent). Also, Laplace=$\Delta = \overrightarrow{\nabla} \cdot \overrightarrow{\nabla}$=div·grad. The change of a function due to a change in coordinates can be written as

$$d\phi = \phi(x_1 + dx_1, x_2 + dx_2, x_3 + dx_3) - \phi(x_1, x_2, x_3).$$

According to the Taylor-formula,

$$\phi(x_1+dx_1, x_2+dx_2, x_3+dx_3) = \phi(x_1, x_2, x_3)+\frac{\partial \phi}{\partial x_1}dx_1+\frac{\partial \phi}{\partial x_2}dx_2+\frac{\partial \phi}{\partial x_3}dx_3+O(h^2).$$

Hence, $d\phi = \overrightarrow{\nabla}\phi \cdot d\overrightarrow{r}$ =grad$\phi \cdot d\overrightarrow{r}$. So, with u=$\phi$

$$\Delta u = \overrightarrow{\nabla} \cdot \overrightarrow{\nabla} u = div \cdot grad u = \overrightarrow{\nabla} \cdot (\overrightarrow{e}_1 \frac{\partial}{\partial x_1} + \overrightarrow{e}_2 \frac{\partial}{\partial x_2} + \overrightarrow{e}_3 \frac{\partial}{\partial x_3})u$$

$$= \overrightarrow{\nabla} \cdot (\frac{\partial \overrightarrow{e}_{x_1}}{h_1}\frac{\partial}{\partial x_1} + \frac{\partial \overrightarrow{e}_{x_2}}{h_2}\frac{\partial}{\partial x_2} + \frac{\partial \overrightarrow{e}_{x_3}}{h_3}\frac{\partial}{\partial x_3})u$$

$$= \frac{1}{h_1 h_2 h_3}[\frac{\partial}{\partial x_1}(\frac{h_2 h_3}{h_1}\frac{\partial u}{\partial x_1}) + \frac{\partial}{\partial x_2}(\frac{h_1 h_3}{h_2}\frac{\partial u}{\partial x_2}) + \frac{\partial}{\partial x_3}(\frac{h_1 h_2}{h_3}\frac{\partial u}{\partial x_3})] \qquad (1.1)$$

In the following we derive the typical FEM formulation of the underlying Poisson problem with respect to the different coordinate systems.

Cartesian coordinates (x,y,z)

$$\overrightarrow{\mathbf{x}} = (x, y, z) \rightarrow \overrightarrow{\mathbf{h}} = (|\frac{\partial \mathbf{x}}{\partial x}|, |\frac{\partial \mathbf{x}}{\partial y}|, |\frac{\partial \mathbf{x}}{\partial z}|) = (1, 1, 1)$$

So Δu is $\frac{\partial u^2}{\partial x^2} + \frac{\partial u^2}{\partial y^2}$ and the FEM formulation is

$$\int (\frac{\partial u}{\partial x}\frac{\partial \varphi}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial \varphi}{\partial y}) \cdot dxdy = \int f\varphi \cdot dxdy,$$

Cylindrical coordinates (ρ, ϕ, z)

$\overrightarrow{\mathbf{x}} = (\rho cos(\phi), \rho sin(\phi), z)$

$\overrightarrow{\mathbf{h}} = (|(cos(\phi), sin(\phi), 0)|, |(-\rho sin(\phi), \rho cos(\phi), 0)|, |(0, 0, 1)|) = (1, \rho, 1)$

So Δu is (symmetric in ϕ)

$$\frac{1}{\rho}(\frac{\partial}{\partial \rho}(\frac{\rho}{1}\frac{\partial u}{\partial \rho}) + \frac{\partial}{\partial z}(\frac{\rho}{1}\frac{\partial u}{\partial z}))$$

and the corresponding FEM formulation writes

$$\int (\frac{\partial u}{\partial \rho}\frac{\partial \varphi}{\partial \rho} + \frac{\partial u}{\partial z}\frac{\partial \varphi}{\partial z}) \cdot \rho d\rho dz = \int f\varphi \cdot \rho d\rho dz$$

Spherical coordinates (ρ, ϑ, ϕ)

$\overrightarrow{\mathbf{x}} = (\rho sin(\vartheta)cos(\phi), \rho sin(\vartheta)sin(\phi), \rho cos(\vartheta))$

$\overrightarrow{\mathbf{h}} = (|(sin(\vartheta)cos(\phi), sin(\vartheta)sin(\phi), cos(\vartheta))|,$

$|(\rho cos(\vartheta)cos(\phi), \rho cos(\vartheta)sin(\phi), -\rho sin(\vartheta))|, |(-\rho sin(\vartheta)sin(\phi), \rho sin(\vartheta)cos(\phi), 0)|)$

$= (1, \rho, \rho sin(\vartheta))$

So Δu is (symmetric in ϕ)

$$\frac{1}{\rho^2 sin(\vartheta)}(\frac{\partial}{\partial \rho}(\frac{\rho^2 sin(\vartheta)}{1}\frac{\partial u}{\partial \rho}) + \frac{\partial}{\partial \vartheta}(\frac{sin(\vartheta)}{1}\frac{\partial u}{\partial \vartheta}))$$

and the FEM formulation reads

$$\int (\frac{\partial u}{\partial \rho}\frac{\partial \varphi}{\partial \rho} + \frac{1}{\rho^2}\frac{\partial u}{\partial \vartheta}\frac{\partial \varphi}{\partial \vartheta}) \cdot \rho^2 sin(\vartheta) d\rho d\vartheta = \int f\varphi \cdot \rho^2 sin(\vartheta) d\rho d\vartheta.$$

One problem for solvers of the resulting linear systems of equations is high refinement. Higher refinement leads to large matrices and many iterative type solvers possess slow convergence. Another problem is the aspect ratio. If the aspect ratio is too high, the coefficients for approximation become small/large and many iterative solvers may not be able to handle such cases. Additionally, a typical problem of spherical coordinates are the metric values $\rho^2 sin(\vartheta)$. Near the z-axis, ρ^2 is very small, the same for $sin(\vartheta)$ near the equator, these have the same effect as examining geometrical aspect ratios. Therefore, in the next chapter we will examine more carefully the characteristics of different solvers for linear systems, all w.r.t. the problems of anisotropies and huge dimensions.

Chapter 2

Numerical components

2.1 Gaussian elimination and CG method

The Gaussian elimination method is a direct solver: it returns the exact solution after analyzing inverse of the matrix involved. Matrices resulting from the discretion of PDEs ('Partial Differential Equations') are often sparse: they have very few non-zero matrix elements. However, during the inversion process the matrix looses its sparsity! For small linear systems this is acceptable and returns an exact solution in short time. However, for larger systems factorization time and storage cost become very expensive.

The Gaussian elimination method above must invert a matrix in order to solve a problem. For huge linear systems this will not only cost a lot of time, but also will require a lot of memory. To avoid this, iterative systems have been proposed. The idea of CG (Conjugate Gradient) methods (see [7]) is to minimize the error in the j'th iteration over affine space defined by K_j=span $\{r_0, Ar_0, ..., A^{j-1}r_0\}$. This is used in many Krylov-space methods. These are iterative methods and theoretically deliver an exact solution in NEQ (number of equations) steps. Unfortunately, the error reduction is often related to $\sqrt{\text{cond(A)}}$ ($\sim$ 1-O(h), h mesh width), so that convergence slows as the grid steps become smaller.

2.2 Fixed-point iterations

For large systems of linear equations, using Gaussian elimination is unrealistic, since the amount of memory and time required to solve it would exceed the limits of many modern computers. For the same reason the matrices are held in sparse format, meaning that only non-zero matrix elements are actually stored. Instead of using direct solvers, fixed-point iteration strategies are employed. They have the form $x^{l+1} = x^l - \omega C^{-1}(Ax^l - b)$. The term in the brackets is called the defect and should converge to zero.

The idea comes out of the following observation:

$$Ax = b \Leftrightarrow Cx = Cx - Ax + b \Leftrightarrow x = x - C^{-1}(Ax - b)$$

The iteration procedure is then

$$x^{l+1} = x^l - C^{-1}(Ax^l - b)$$

$$= (I - C^{-1}A)x^l + C^{-1}b =: Bx^l + c.$$

The error is then $e^{l+1} := x^{l+1} - x = Bx^l + c - (Bx + c) = Be^l$. In order for the defect to convergence to zero, a necessary requirement is $||B|| <1$.

The initial matrix A is preconditioned with a matrix C, in the hope that the convergence of the iteration system improves. The convergence is directly related to its condition, $|\frac{\lambda_{max}}{\lambda_{min}}|$, and the strategy is for cond(C^{-1}A) <cond(A). Also, C^{-1} should be quick and easy to find. Often, a parameter ω is employed. For example, for C=I, if $\omega \leq \frac{1}{\lambda_{max}(A)}$ then $||I - \omega C^{-1}A|| < 1$. These preconditioners are employed in the previous fixed-point iterations and also in conjugate-gradient-like methods for high-frequency damping.

Let A=L+D+U, with L=lower-, D=diagonal-, and U=upper-parts of A. The following typical choices for C are taken. The short characterization of the resulting methods is based on the numerical studies in the following chapter.

1. Classical Richardson: C=I, $\omega \leq \frac{1}{\lambda_{max}(A)}$

2. Jacobi: C=D, $\omega \geq 0$: The Jacobi preconditioner works best for isotropic grids. Dirichlet/cartesian or Neumann/cylindrical situations are preferable. It has optimal convergence for diagonal matrices. It has the advantage that it requires very little extra memory (or none at all!). It converges very slowly like 1-$O(h)$ for grid-step h.

3. Gauß-Seidel/SOR: C=ωL+D, $\omega \in (0, 2)$: Gauß-Seidel/SOR behaves similar like Jacobi: Dirichlet/cartesian or Neumann/cylinder situations are preferable. This preconditioner works optimal for lower triangular matrices arising from convection-dominated problems: -$\varepsilon\Delta$U+$\beta\nabla$U (downwind numbering). GS (Gauß-Seidel) converges also like 1-$O(h)$ for grid-step h.

4. TRI: C=tridiagonal part of A: The TRI (tridiagonal) preconditioners are chosen because of the idea that when one refines much stronger into one direction, the corresponding FEM stiffness matrix converges to a triangular matrix if appropriate renumbering is used. Therefore, the TRI-based schemes are good candidates for large aspect ratios.

5. ADI: C=alternates between the tridiagonal parts of A for row- and columnwise numbering which may be necessary for anisotropic refinements in both directions. A routine that 'virtually' transposes the grid is used to fetch the correct values from the main matrix and also to permute the vectors accordingly.

6. GSTRI/ADI: The GSTRI preconditioner is a combination of the Gauß-Seidel and tridiagonal preconditioners. It uses lower part of the main matrix plus the diagonal plus the superdiagonal. One would expect this method to have the advantages of both the TRI and the GS preconditioners. Since it can be interpreted as linewise Gauß-Seidel, this method is also easy to implement, works with high computational performance, and should lead to uniformly bounded and excellent convergence rates on isotropic as well as anisotropic meshes. These schemes tend to be our favorites!

7. ILU: C=incomplete LU factorization of A (LU without fill-in), $\omega \geq 0$: The ILU preconditioner is (often) the most robust method of all. Per definition, it avoids fill-in and is very robust with respect to anisotropic meshes and convection dominated problems. However, the optimal

choice of ω is not clear, and the computational costs are higher than for the previous schemes.

The direct (Gaussian) solver is based on an implementation of Lapack routines DGBTRF and DGBTRS.Since the single grid LU method results in 'exact' solutions independent from aspect ratios, one could suppose that we could forget the rest. However, it has its disadvantages. The exact LU solver for solving on one level has the following cpu-times in seconds.

LU, single grid, Cartesian, isotropic				
level	unknowns	storage (RAM)	factorization time	solve time
5	1089	1.1Mb	4.57-2	5.24-3
6	4225	7.5Mb	4.78-1	5.55-2
7	16641	54.4Mb	5.32-0	4.01-1
8	66049	414.1Mb	6.67+1	3.17+0

The time for solving increases by a factor of 7-8, factorization time increases by 10-12, while the number of unknowns increases only by 4 for each level. From the table, one can infer that level 9 is not only slow with respect to factorization time, but also most likely not possible because of memory requirements.

On the other hand, typical multigrid times and convergence rates are shown in the next table. From these results, it can be immediately seen that the factorization time for the exact LU factorization outweighs any benefits gained from the ILU method.

MG(ILU), Cartesian, isotropic, smooth=2, eps=1e-8			
NLMAX	conv.(Iter)	storage	cpu-time
5	.20-1 (6)	0.4Mb	4.0-2
6	.20-1 (6)	1.2Mb	2.1-1
7	.19-1 (6)	4.7Mb	1.2+0
8	.19-1 (6)	18.5Mb	6.6+0
9	.18-1 (6)	73.5Mb	3.0+1

The next tables show the effects of different stopping criteria and of increased smoothing for MG(ILU). Convergence rate excelled whereas the time needed decreased slightly.

MG(ILU), Cartesian, isotropic, smooth=2,eps=1e-2			
NLMAX	conv.(Iter)	storage	cpu-time
5	.28-2 (1)	0.4Mb	1.0-2
6	.27-2 (2)	1.2Mb	8.0-1
7	.24-2 (3)	4.7Mb	4.8-1
8	.20-2 (2)	18.5Mb	2.3+0

MG(ILU), Cartesian, isotropic, smooth=4,eps=1e-2			
NLMAX	conv.(Iter)	storage	cpu-time
5	.14-2 (1)	0.4Mb	2.0-2
6	.87-3 (1)	1.2Mb	7.0-1
7	.59-3 (1)	4.7Mb	3.8-1
8	.42-3 (1)	18.5Mb	2.1+0

As one can see, the direct LU method has the advantage that it returns an 'exact' result in similar times on lower levels, but on higher levels, the multigrid algorithm has timing and storage advantages.

In the following, the average timings of the other preconditioners as smoothers are presented. For an isotropic grid, the following preconditioners have the following timings for one multigrid iteration on Dirichlet boundary conditions, 2 smoothing steps, F-cycle. The table shows, that ILU takes the longest time to complete one multigrid step. We have to remark, that the implementation of the matrix-vector (MV) operations is not based on the optimized FEAST components but on the described sparse techniques such that further enormous speed-up, at least for TRI and GSTRI, can be expected.

CPUtime	5	6	7	8
JAC	4.17-3	1.62-2	9.26-2	6.61-1
TRI	6.97-3	3.00-2	1.60-1	9.17-1
GS	7.20-3	3.29-2	1.74-1	9.86-1
GSTRI	8.00-3	3.11-2	1.71-1	9.84-1
ILU	9.82-3	5.01-2	2.31-1	1.14+0

Next, the preconditioned CG-method is examined for solving problems on one single mesh, for globally constant aspect ratios.

The following tables are created for isotropic refining while adjusting the rightmost x-coordinates which correspond to aspect ratios 1,10,100,1000 respectively. The TRI preconditioner matrix gets closer to the exact inverse of the main matrix as the xmax declines.

We perform CG with different preconditioning to test the sensitivity of CG w.r.t. such mesh anisotropies.

CG(TRI), Dirichlet, Cartesian, 9-star				
level	x=1.0	x=0.1	x=0.01	x=0.001
				
5	.67 (47)	.30 (16)	.17 (11)	.15 (10)
time	.051	.02	.016	0.014
6	.82 (92)	.56 (32)	.16 (10)	.13 (9)
time	.331	.11	.045	.037
7	.90 (183)	.75 (66)	.15 (10)	.12 (9)
time	3.73	1.36	.249	.219
8	div	.87 (132)	.19 (12)	.97-1 (8)
time		13.7	1.39	1.07

CG(ILU), Dirichlet, Cartesian				
level	x=1.0	x=0.1	x=0.01	x=0.001
5	.43 (22)	.37 (19)	.56-2 (4)	.46-4 (2)
time	.035	.031	.01	.007
6	.62 (39)	.59 (37)	.39-1 (6)	.11-3 (3)
time	.214	.203	.044	.025
7	.77 (72)	.77 (73)	.14 (10)	.37-3 (3)
time	2.37	2.63	.350	.123
8	.87 (141)	.88 (147)	.36 (19)	.12-2 (3)
time	20.1	22.6	3.28	.553

Above, it can be immediately seen that convergence rates decline with increasing level: this is due to the 1-O(h) behavior of Krylov methods. On the other hand, for large aspect ratios they perfectly converge since the preconditioner gets optimal. Below we show that the TRI and ILU preconditioners in multigrid require about the same number of iterations for levels 5 through 8, and that their behavior improves for large aspect ratios.

MG(TRI), Dirichlet, Cartesian, smooth=2, 5-star				
Level	x=1.0	x=0.1	x=0.01	x=0.001
5	.12 (10)	.36-2 (4)	.10-3 (3)	.10-3 (3)
time	.04	.02	.02	.02
6	.13 (10)	.11-1 (6)	.10 (3)	.10-3 (4)
time	.22	.12	.06	.08
7	.13 (11)	.11-1 (6)	.10-3 (3)	.10-3 (4)
time	1.11	.68	.32	.41
8	.13 (11)	.11-1 (6)	.40-3 (4)	.10-3 (4)
time	6.43	3.62	2.46	2.40

MG(TRI), Dirichlet, Cartesian, smooth=2, 9-star				
Level	x=1.0	x=0.1	x=0.01	x=0.001
5	.52-1 (7)	.18-1 (6)	.25-1 (7)	.30-1 (8)
time	.04	.03	.04	.04
6	.50-1 (7)	.30-1 (7)	.21-1 (7)	.25-1 (8)
time	.16	.16	.16	.19
7	.48-1 (7)	.18-1 (6)	.17-1 (7)	.21-1 (8)
time	.92	.80	.91	1.07
8	.45-1 (7)	.14-1 (6)	.10-1 (6)	.18-1 (8)
time	5.81	4.97	5.09	6.45

MG(ILU), Dirichlet, Cartesian, smooth=2, 9-star				
Level	x=1.0	x=0.1	x=0.01	x=0.001
5	.38-2 (4)	.33-2 (4)	.20-11 (1)	.17-17 (1)
time	.03	.03	.01	.01
6	.38-2 (4)	.78-2 (5)	.23-6 (2)	.30-17 (1)
time	.14	.17	.02	.04
7	.37-2 (4)	.81-2 (5)	.11-2 (4)	.75-17 (1)
time	.84	1.01	.81	.21
8	.34-2 (4)	.12-1 (6)	.48-2 (5)	.33-16 (1)
time	4.48	5.33	6.29	1.19

From the above tables, it is inferred that the MG algorithm converges much faster than the one-grid method, particularly on fine meshes. Also, for aspect-ratios significantly larger than 1, that the 5-Star TRI converges better than the 9-Star TRI and for aspect-ratio of size 1 the 9-Star TRI does better.

The refinement method used in the following tests produces a tensorproduct grid with elements with gradually higher aspect ratios as one moves toward the left side.

We start with the CG methods.

Single-Grid, CG(TRI), Dirichlet, Cartesian			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
			
5	.66 (47)	.73 (62)	.72 (58)
time	.06	.08	.10
6	.81 (92)	.85 (112)	.85 (115)
time	.29	.32	.33
7	.90 (183)	.92 (242)	.92 (225)
time	3.82	5.59	5.27
8	(>300)	(>300)	(>300)

Single-Grid, CG(ILU), Dirichlet, Cartesian					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.43 (22)	.39 (21)	.33 (17)	.38 (20)	.28 (16)
6	.62 (39)	.61 (39)	.56 (33)	.62 (39)	.54 (31)
7	.77 (72)	.78 (76)	.75 (67)	.79 (82)	.74 (63)
8	.87 (141)	.88 (157)	.87 (143)	.89 (172)	.86 (130)
9	.93 (278)	.95 (300)	.93 (286)	(>300)	.93 (286)

MG, Dirichlet, Cartesian, smooth=2, 5-star					
	TRI			ADI	
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.11 (9)	.24 (14)	.35 (19)	.13 (10)	.11 (9)
time	.04	.07	.08	.06	.05
6	.11 (10)	.24 (15)	.36 (20)	.09 (9)	.12 (10)
time	.23	.33	.43	.22	.23
7	.10 (10)	.24 (15)	.35 (20)	.09 (9)	.12 (10)
time	1.06	1.57	2.15	1.08	1.19
8	.10 (10)	.24 (16)	.35 (21)	.09 (9)	.11 (10)
time	6.05	9.61	12.52	5.39	5.76

MG, Dirichlet, Cartesian, smooth=2, 9-star					
	TRI			ADI	
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.52-1 (7)	.13+0 (10)	.23+0 (14)	.87-1 (8)	.93-1 (8)
time	.04	.06	.08	.04	.04
6	.50-1 (7)	.13+0 (10)	.23+0 (14)	.10+0 (9)	.10+0 (9)
time	.16	.29	.40	.21	.26
7	.48-1 (7)	.13+0 (11)	.22+0 (14)	.11+0 (10)	.14+0 (11)
time	.97	1.71	2.16	1.35	1.58
8	.45-1 (7)	.13+0 (11)	.22+0 (15)	.14+0 (11)	.20+0 (14)
time	5.77	9.69	13.62	9.38	11.12

The above tables show the results for variable local aspect-ratios with ILU.

MG, ILU, Dirichlet, Cartesian, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	0.38-2 (4)	0.30-2 (4)	0.25-2 (4)	0.39-2 (4)	0.46-2 (4)
time	.03	.03	.03	.03	.03
6	0.38-2 (4)	0.38-2 (4)	0.53-2 (4)	0.57-2 (4)	0.23-1 (6)
time	.16	.16	.16	.16	.23
7	0.37-2 (4)	0.42-2 (4)	0.12-1 (5)	0.82-2 (5)	0.26-1 (6)
time	.83	.83	1.06	1.04	1.25
8	0.34-2 (4)	0.79-2 (5)	0.24-1 (6)	0.22-1 (6)	0.28-1 (10)
time	4.75	5.89	14.03	7.04	11.66

Both 9-point (FEM) and 5-point (FD) discretions can be optimally treated by multigrid while the 'one-level' CG gets into trouble for fine meshes. Since Gaussian elimination as well as CG methods have problems with such large meshes, multigrid is our preferred solver and will be discussed in the next section. On the other hand, the numerical results show that convergence rates are not the optimal measure (compare ILU vs. ADI). The aspect of optimal implementation techniques is discussed in section 1.1.

2.3 Multigrid algorithms

The following multigrid routine needs a matrix-vector routine and a preconditioner routine and several control parameters. Then, our MG algorithm MG(.,.,.) reads:

MG(l,u_l^0,g_l) algorithm:

For problem $A_l x = g$ let l=level, S_l=smoother, P_{l-1}^l=Prolongation, R_l^{l-1}=Restriction, and u_l^0=start solution.

If l=1, MG(l,u_1^0,g_1) returns $A_1^{-1}g$

If l$\geq$1 perform the following steps

1. m-presmoothing steps:
   - $u_l^m = S_l u_l^{m-1} = \ldots = S_l^m(u_l^0)$
   - that means m-steps with JAC, GS, ILU, TRI, ADI, TRIGS or ADIGS
2. Coarse grid correction:
   - Restrict the defect $d_l = g_l - A_l\, u_l^m$, that means $g_{l-1} = R_l^{l-1} d_l$, and solve recursively:
   - u_{l-1}^i = MG(l-1, u_{l-1}^{i-1}, g_{l-1}) i=1,..,p, $u_{l-1}^0 = 0$
   - $u_l^{m+1} = u_l^m + P_{l-1}^l u_{l-1}^p$
3. n - postsmoothing steps:
   - $u_l^{m+n+1} = S_l^n(u_l^{m+1})$
   - Finally set MG(l,u_l^0,g_l) := u_l^{m+n+1}

Prolongation and Restriction operators for the grids are required which can be implemented via DAXPY-like operations.

Prolongation from coarse-grids to finer grids is typically realized through the following interpolation stencil; a node at the center is added to its neighbors with the following weights on tensorproduct meshes:

0.25	0.5	0.25
0.5	1.0	.0.5
0.25	0.5	0.25

The corresponding restriction operator restricts defects from fine grids onto coarser grids. It is realized through the next stencil, here the neighboring nodes are summed-up, using the weights in the stencil, onto the center node. It is the adjoint of the prolongation operator: the transposed prolongation matrix is equal to the restriction matrix.

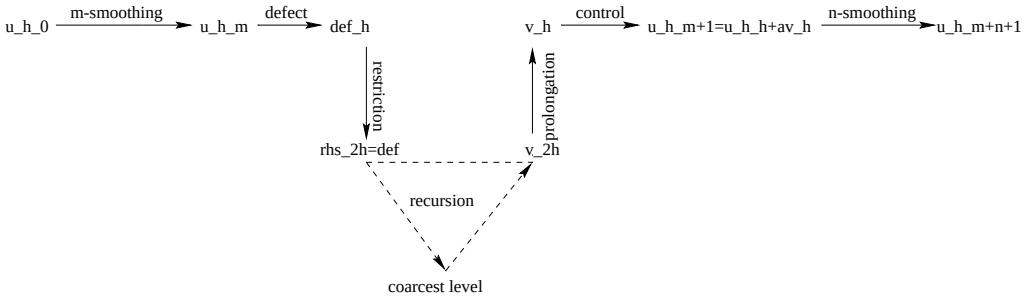
These operators are, of course valid for uniformly refined spaces. The real weights are derived from corresponding bilinear interpolation. But since out tests on locally anisotropic meshes have shown that in such locally refined meshes, the smoother is very strong, the standard 'isotropic' grid transfer seems to be sufficiently efficient.

The amount of extra memory required by each preconditioner can be seen in the table below. NEQ is the number of unknowns for each level, NA=number of non-zero elements in matrix. All values in the table are to be summed over the levels used and correspond to standard implementation.

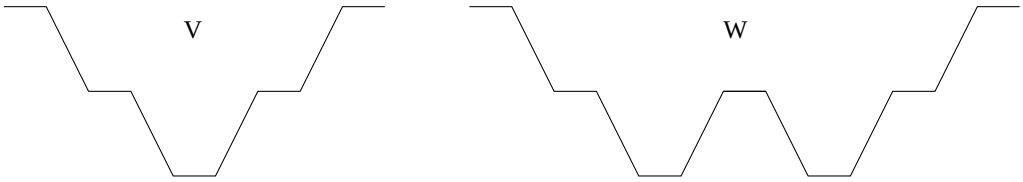
Jac	-
GS	-
TRI	3*NEQ
ADI	2*TRI
GSTRI	1*TRI
GSADI	2*TRI
ILU	NA

As explained above, the multigrid algorithm is composed of several steps: smoothing, restriction, solving and prolongation. The orders in which the various levels appear are suggestive for the name of the cycle-type (V, W, F)

Here is a figure of the V-cycle:



In the W-cycle, after the coarsest level, the MG level does not immediately go completely to the finest level, but instead goes a level upward, then downward, and the towards the finest level. This movement has a 'W' appearance. The F-cycle can be viewed as an intermediate combination of V- and W-cycles.



From the table the F- and W-cycles show the best rates, the V-cycle behaves sometimes worse:

Dirichlet, MG(JAC), isotropic, epsmg=1d-8			
levels	F	V	W
5	.84-1 (8)	.84-1 (8)	.84-1 (8)
6	.83-1 (9)	.10 (9)	.83 (9)
7	.80-1 (9)	.11 (10)	.80-1 (9)
8	.77-1 (9)	.12 (11)	.77-1 (9)
9	.74-1 (9)	.12 (11)	.74-1 (9)

In the following chapter, we vary the choice of smoother: in this work several smoothers are tested and tabled. They are Jacobi, Gauß-Seidel, Tridiagonal, ADI, ILU and GSTRI. The ADI method here is different from the traditional

method: In even steps we apply the tridiagonal preconditioner from rowwise numbering while in odd steps the matrix from the columnwise numbering is taken. The GSTRI method used here is the GS method with an additional super-one diagonal, that means linewise Gauß-Seidel scheme.

Although the ILU method seems to possess the best convergence behavior for many situations, the factorization-timing properties are very expensive for large scale jobs, and the storage costs are large. Therefore, the aim of the following numerical studies is the search for good smoothers with respect to good numerical and computational behavior, and this for several mesh aspect ratios on the different coordinate systems.

Chapter 3

Numerical studies

3.1 Evaluation on orthogonal quadrilaterals

For all tests a zero start-solution vector (besides the boundary components) is employed. The choice of damping parameter ω is critical for the optimal performance of the basic iteration. This value often depends not only upon the refinement parameters used, but also on the actual grid level. Thus choosing an optimal a-priori ω-value is not always possible for all smoothers.

Tests were made using the described multigrid algorithm to gain 8 digits accuracy w.r.t. the starting residual and usually two pre- and post-smoothing steps were made. The problem to be solved in the tests was $-\Delta u = f$. The exact solution is $u = 16x(1-x)y(1-y)$. The right hand side is then $32(x(1-x)+y(1-y))$. This problem has homogeneous boundary values on the $[0,1]\times[0,1]$ domain boundary which is used for the first set of tests. However, since we are mainly interested in the multigrid convergence behavior, the underlying exact solution does not matter.

3.1.1 Tests on the Jacobi preconditioner

Using Dirichlet boundary conditions for cartesian and cylindrical coordinate systems it was found from initial tests that ω_{opt} is often about ω=0.7 for the

(.5,1,1) configuration (see section 1.2 for the description of the refinement parameters: (0.5,1.0,1.0) corresponds to an isotropic mesh with equidistant refinement) and about ω=0.6 for the other configurations. The following tables show the convergence rates and the resulting number of multigrid cycles. The refinement parameters are explained in section 1.2: (0.5,0.5,1.0) denotes a moderately anisotropic mesh, while (0.25,0.25,0.5) corresponds to very significant local mesh distortions. $()^2$ means refinement in both directions. See section 1.2 for corresponding figures. Here, we calculate on a unit square, 'smooth' denotes the number of smoothing steps.

MG(JAC), Dirichlet, Cartesian, smooth=4					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5, .5, 1)^2$	$(.25, .25, .5)^2$
5	.28-1 (7)	.34 (19)	.70 (56)	.46 (26)	.80 (86)
6	.27-1 (8)	.49 (29)	.85 (130)	.62 (43)	(>150)
7	.26-1 (8)	.66 (51)	(>150)	.76 (79)	(>150)
8	.24-1 (8)	.79 (94)	(>150)	.86 (149)	(>150)

MG(JAC), Dirichlet, Cartesian, smooth=8					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5, .5, 1)^2$	$(.25, .25, .5)^2$
5	.16-1 (5)	.13 (10)	.50 (8)	.22 (13)	.64 (44)
6	.15-1 (5)	.25 (15)	.73 (66)	.40 (22)	.83 (111)
7	.14-1 (5)	.44 (26)	.87 (151)	.59 (40)	.92 (260)
8	.15-1 (6)	.63 (47)	(>300)	.75 (75)	(>300)

The Jacobi preconditioner did not converge for Dirichlet or Neumann boundary conditions when spherical coordinates were used. This is due to the metric-coefficients that build the matrix; they have the same effect as adding geometrical aspect-ratios, which the Jacobi and Gauß-Seidel preconditioners badly approximate. Additionally, Jacobi smoothing deteriorates dramatically with increasing the aspect ratios, in particular depending on the problem size.

The above tables show the effect of the number of smoothing operations, too. Smoothing improves convergence, but on the other hand smoothing takes up the largest percentage of the total solving time, thus increasing smoothing operations improves the performance of the preconditioner, but also increases the total time required for the multigrid solver.

In the following, we try to adapt the number of necessary smoothing steps to

guarantee a multigrid convergence of about ρ_{MG}=0.5. As can be seen, this number is significantly dependent on the mesh level used for non-isotropic meshes: it has to be doubled or even more if the mesh is refined. 'Time' denotes the elapsed CPU time.

MG(JAC), Dirichlet, Cartesian					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.61-1 (7)	.34 (19)	.50 (28)	.46 (26)	.46 (25)
smooth	2	4	8	4	14
AR	1	16	1024	24	1920
Time	3.3-2	0.12	0.30	0.17	0.42
6	.61-1 (8)	.49 (29)	.49 (28)	.50 (29)	.51 (30)
smooth	2	4	19	6	30
AR	1	32	4096	48	7680
Time	2.9-2	0.66	2.24	0.86	4.97
7	.58-1 (8)	.49 (30)	.50 (30)	.49 (29)	.46 (27)
smooth	2	7	41	11	80
AR	1	64	16384	96	30720
Time	0.75	6.84	38.49	9.86	61.95
8	.56-1 (8)	.50 (32)	.57 (38)	.47 (29)	.43 (25)
smooth	2	12	81	21	200
AR	1	128	65536	192	122880
Time	5.64	96.02	703.83	148.26	1311.82

As can be seen, the increase in CPU time is more than a factor of 4 from level to level, as could be predicted by the performance measurements for sparse matrix-vector applications in Section 1.

An interesting observation is that for Neumann boundary conditions at the left (and bottom) edges, convergence rates significantly deteriorate! This can be explained by theoretical results of Yserentant ([15]) who has analyzed the behavior of Jacobi preconditioners on locally refined meshes toward the boundary.

MG(JAC), Neumann, Cartesian					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.10 (9)	.46 (26)	.56 (34)	.45 (25)	.56 (34)
smooth	2	16	660	30	1320
Time	4.24-2	0.34	22.50	0.82	45.44
6	.11 (10)	.47 (27)	(>300)	.50 (29)	(>300)
smooth	2	35		60	
Time	0.16	3.92		7.51	
7	.11 (10)	.52 (32)	(>300)	.54 (34)	(>300)
smooth	2	70		120	
Time	0.90	62.12		144.25	
8	.11 (10)	.53 (35)	(>300)	(>300)	(>300)
smooth	2	150			
Time	6.64	1242.95			

Another grid was designed with refinements made toward an inner layer. For this grid 'Dirichlet2', smoothing operations have to be increased to reach similar convergence results. As seen below convergence rates for the anisotropic versions of this grid are very bad when one takes into account the number of smoothing steps used.

MG(JAC), Dirichlet2, Cartesian					
Level	(.5,1,1) 	(.5,.5,1) 	(.25,.25,.5) 	$(.5,.5,1)^2$ 	$(.25,.25,.5)^2$ 
5	.61-1 (7)	.66 (48)	.45 (24)	.49 (28)	.69 (53)
smooth	2	4	300	12	265
6	.61-1 (8)	.50 (30)	.67 (51)	.54 (33)	(>300)
smooth	2	16	600	24	
7	.58-1 (8)	.50 (31)	(>300)	.54 (34)	(>300)
smooth	2	36		54	
8	.56-1 (8)	.51 (33)	(>300)	.46 (28)	(>300)
smooth	2	80		154	

From these tests, one can easily deduce where a preconditioner will have difficulties by examining the number of smoothing steps required to reach a desired convergence rate.

Next, we examine cylindrical coordinates.

MG(JAC), Dirichlet, Cylinder, AR as above					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.18 (12)	.35 (19)	.50 (29)	.49 (28)	.49 (28)
smooth	2	4	12	4	20
Time	5.50-2	0.12	0.43	0.18	0.64
6	.22 (14)	.53 (33)	.48 (29)	.54 (34)	.51 (31)
smooth	2	4	30	6	48
Time	0.23	0.77	3.41	1.05	5.78
7	.25 (16)	.49 (31)	.55 (36)	.54 (35)	.55 (36)
smooth	2	8	60	11	100
Time	1.51	7.97	60.98	12.32	101.53
8	.28 (18)	.51 (33)	.65 (51)	.49 (31)	.60 (43)
smooth	2	14	100	24	200
Time	11.78	115.36	1210.40	178.43	2252.57

The results for Dirichlet boundary conditions are the same as for cartesian coordinates. Only for the isotropic case (0.5,1.0,1.0), Dirichlet calculations behave better due to the isotropic matrix coefficients now. In contrast, Neumann boundary conditions behave better in computations with cylindrical coordinates as the following table shows: fortunately, this is the physically more correct situation.

MG(JAC), Neumann, Cylinder					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.55-1 (7)	.46 (26)	.50 (29)	.50 (29)	.55 (33)
smooth	2	5	20	26	1320
Time	3.41-2	0.19	0.66	0.84	43.94
6	.56-1 (8)	.47 (28)	.51 (31)	.49 (30)	(>300)
smooth	2	8	42	60	
Time	0.13	1.06	5.18	7.25	
7	.54-1 (9)	.46 (28)	.56 (37)	.53 (34)	(>300)
smooth	2	14	82	120	
Time	0.74	11.84	86.16	115.25	
8	.52-1 (9)	.52 (34)	(>300)	(>300)	(>300)
smooth	2	20			
Time	5.32	170.01			

That means, there is a significant difference for the multigrid w.r.t. Dirichlet or Neumann boundary conditions. Dirichlet behaves often better, especially for high local aspect ratios if the mesh is strongly refined near the boundary. Similar effects are discussed in [15] which contains a theoretical background.

Next, isotropic refinement in the x-direction is used and x_max is adjusted so that the aspect-ratio is globally constant. First, we show the results on levels 5-7 for cartesian coordinates, followed by the corresponding results for the cylindrical case. The next table shows the results from computations using ω=0.6 and ω=0.7. It can be immediately seen that the results are sensitive of the choice for ω.

MG(JAC), Dirichlet, Cartesian, smooth=2						
Level	AR=1	AR=2	AR=3	AR=4	AR=5	AR=10
						
5	.84-1	.29	.53	.67	.76	.91
6	.83-1	.27	.51	.66	.76	.92
7	.80-1	.25	.48	.64	.74	.92
5	.61-1	.23	.55	.86	(>300)	(>300)
6	.61-1	.21	.56	.88	(>300)	(>300)
7	.58-1	.18	.56	.88	(>300)	(>300)

MG(JAC), Dirichlet, Cartesian, smooth=8						
Level	AR=1	AR=2	AR=3	AR=4	AR=5	AR=10
5	.16-1	.16-1	.88-1	.21	.34	.69
6	.15-1	.13-1	.79-1	.20	.34	.72
7	.14-1	.10-1	.60-1	.17	.31	.71
5	.13-1	.13-1	.99-1	.55	(>300)	(>300)
6	.13-1	.10-1	.10+0	.60	(>300)	(>300)
7	.12-1	.84-2	.10+0	.60	(>300)	(>300)

MG(JAC), Neumann, Cartesian, smooth=2						
Level	AR=1	AR=2	AR=3	AR=4	AR=5	AR=10
5	.14	.29	.52	.67	.76	.92
6	.15	.27	.50	.66	.75	.92
7	.15	.25	.47	.64	.74	.91
5	.10	.23	.56	.88	(>300)	(>300)
6	.11	.21	.56	.88	(>300)	(>300)
7	.11	.20	.56	.88	(>300)	(>300)

MG(JAC), Neumann, Cartesian, smooth=8						
Level	AR=1	AR=2	AR=3	AR=4	AR=5	AR=10
5	.31-1	.17-1	.75-1	.21	.34	.72
6	.31-1	.14-1	.68-1	.19	.32	.72
7	.30-1	.11-1	.59-1	.17	.30	.71
5	.29-1	.13-1	.10+0	.60	(>300)	(>300)
6	.28-1	.11-1	.10+0	.60	(>300)	(>300)
7	.28-1	.95-2	.10+0	.61	(>300)	(>300)

The tables show that aspect ratios greater than 2 lead already to problems for Jacobi smoothing. Otherwise, the number of smoothing steps has to be significantly increased. Again, Dirichlet boundary conditions behave slightly better than Neumann boundary conditions, but the difference is much smaller than compared with local anisotropies.

We come the the following conclusions:

- **aspect ratio:** Jacobi behaves well for aspect ratios ≤ 2, otherwise much more smoothing steps are required, depending on mesh level.
- **boundary conditions:** Dirichlet boundary conditions behave better for cartesian coordinates especially for very local anisotropies, while there is no big difference for cylindrical systems.
- **coordinate systems:** Cartesian and cylindrical behave similar, spherical coordinates are impossible since the matrix coefficients correspond to large geometrical aspect ratios.

Use Jacobi only for very isotropic meshes (∼squares)!!!

3.1.2 Tests on the Gauß-Seidel preconditioner

We start with the same tests as for the JAC smoother. Again, there are often large problems for spherical coordinates.

MG(GS), Dirichlet, Cartesian, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.23-1 (6)	.17+0 (12)	.57+0 (35)	.29+0 (16)	.70+0 (54)
6	.22-1 (6)	.32+0 (18)	.78+0 (81)	.47+0 (27)	.86+0 (135)
7	.21-1 (6)	.52+0 (32)	(>150)	.66+0 (51)	(>150)
8	.20-1 (6)	.69+0 (58)	(>150)	.80+0 (97)	(>150)

MG(GS), Dirichlet, Cylinder, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.96-1 (9)	.18+0 (12)	.67+0 (51)	.31+0 (17)	.79+0 (84)
6	.12+0 (10)	.35+0 (20)	.84+0 (125)	.50+0 (30)	(>150)
7	.14+0 (12)	.55+0 (37)	(>150)	.69+0 (57)	(>150)
8	.17+0 (13)	.72+0 (69)	(>150)	.82+0 (110)	(>150)

In order to reach convergence rates near ρ_{MG}=0.5, the number of smoothing operations has to be increased. On anisotropic meshes, the following table for Cartesian coordinates demonstrates this, similar results could by obtained for cylindrical and spherical coordinates.

MG(GS), Dirichlet, Cartesian, xmax=1					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.23-1(6)	.17 (12)	.33 (18)	.29 (16)	.49 (27)
smooth	2	2	4	2	4
6	.22-1 (6)	.32 (18)	.48 (28)	.47 (27)	.49 (28)
smooth	2	2	6	2	10
7	.21-1 (6)	.28 (17)	.46 (27)	.43 (25)	.53 (32)
smooth	2	4	14	4	20
8	.20-1 (6)	.48 (30)	.53 (34)	.41 (24)	.53 (38)
smooth	2	4	28	8	40

As before, the number of smoothing steps has to be doubled for each further mesh refinement if large aspect ratios occur. However, Gauß-Seidel behaves clearly better than the Jacobi smoother.

Non-isotropic grids and Neumann boundary conditions proved again to be more difficult, also for the GS preconditioner, especially when Cartesian coordinates were employed. As before, Neumann boundary conditions behave better for cylindrical coordinates which are physically more suited.

MG(GS), Cartesian, Neumann, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.45-1 (7)	(>300)	(>150)	.85+0 (125)	(>150)
6	.45-1 (7)	(>300)	(>150)	(>150)	(>150)
7	.44-1 (7)	(>300)	(>150)	(>150)	(>150)
8	.44-1 (7)	(>300)	(>150)	(>150)	(>150)

MG(GS), Cylinder, Neumann, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.28+0 (16)	.45+0 (25)	.79+0 (85)	.85+0 (125)	(>150)
6	.41+0 (24)	.63+0 (46)	(>150)	(>150)	(>150)
7	.55+0 (37)	.77+0 (82)	(>150)	(>150)	(>150)
8	.68+0 (59)	(>150)	(>150)	(>150)	(>150)

Again, we show in the following tables the number of necessary smoothing steps to reach convergence rates of about $\rho \sim 0.5$ which has to be compared with the results for the cartesian coordinates. As discussed before, the increase of CPU-time is more than linear, due to cache and memory access problems of the underlying implementation (see Section 1).

MG(GS), Neumann(l), Cylinder, xmax=1			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
5	.23-1 (6)	.33 (20)	.51 (30)
smooth	2	2	6
Time	4.13-2	0.13	0.38
6	.22-1 (6)	.55 (35)	.54 (34)
smooth	2	2	12
Time	0.17	0.99	4.26
7	.21-1 (6)	.49 (30)	.58 (39)
smooth	2	4	24
Time	0.87	7.67	52.06
8	.20-1 (6)	.54 (36)	(>300)
smooth	2	6	
Time	5.54	84.96	

Next, isotropic refinement in the x-direction is used and xmax is adjusted so that the aspect-ratio is globally constant.

Level	AR=1	AR=2	AR=3	AR=4	AR=5	AR=10
						
MG(GS), Dirichlet, Cartesian, smooth=2						
5	.23-1	.34-1	.14	.28	.42	.73
6	.22-1	.35-1	.13	.27	.41	.76
7	.21-1	.35-1	.10	.24	.38	.75
MG(GS), Dirichlet, Cylinder, smooth=2						
Level	AR=1	AR=2	AR=3	AR=4	AR=5	AR=10
5	.98-1	.47-1	.12	.27	.41	.75
6	.12	.56-1	.11	.26	.40	.77
7	.14	.57-1	.10	.25	.39	.76

MG(GS), Neumann, Cartesian, smooth=2						
Level	AR=1	AR=2	AR=3	AR=4	AR=5	AR=10
5	.42-1	.34-1	.14	.30	.43	.77
6	.42-1	.35-1	.13	.27	.41	.77
7	.42-1	.35-1	.10	.25	.38	.76

MG(GS), Neumann, Cylinder, smooth=2						
Level	AR=1	AR=2	AR=3	AR=4	AR=5	AR=10
5	.23-1	.32-1	.12	.27	.41	.76
6	.22-1	.33-1	.11	.26	.40	.77
7	.21-1	.33-1	.10	.25	.39	.76

MG(GS), Neumann, Spherical, smooth=2						
Level	AR=1	AR=2	AR=3	AR=4	AR=5	AR=10
5	.10	.94-1	.18	.32	.44	.76
6	.12	.11	.22	.35	.47	.78
7	.15	.11	.23	.37	.49	.79

MG(GS), Dirichlet, Cartesian, smooth=8						
Level	AR=1	AR=2	AR=3	AR=4	AR=5	AR=10
5	.44-2	.28-2	.43-2	.11-1	.37-1	.30
6	.44-2	.25-2	.39-2	.10-1	.36-1	.34
7	.42-2	.19-2	.30-2	.86-2	.30-1	.33

MG(GS), Dirichlet, Cylinder, smooth=8						
Level	AR=1	AR=2	AR=3	AR=4	AR=5	AR=10
5	.37-1	.18-1	.17-1	.16-1	.32-1	.32
6	.56-1	.21-1	.19-1	.18-1	.31-1	.35
7	.74-1	.22-1	.24-1	.23-1	.27-1	.34

We come to the conclusions for the Gauß-Seidel smoother:

- **aspect ratios:** Works fine up to aspect ratio 5, otherwise more smoothing steps are needed, depending on the mesh level.

- **boundary conditions:** Dirichlet boundary conditions behave similar for cartesian and cylindrical, the combination Neumann/cylindrical is slightly better than Neumann/cartesian.

- **coordinate systems:** Cartesian and cylindrical behave similar for Dirichlet boundary conditions, spherical coordinates work only for Neumann conditions.

Use GS for slightly anisotropic meshes (AR∼5)!!!

3.1.3 Tests on the TRI- and ADI-preconditioners

The TRI-preconditioner possesses very good convergence rates on highly anisotropic meshes: then, the preconditioner tends to be an exact solver! Damping values for ω range from 0.7, for isotropic configurations, to 1.0 for extreme aspect-ratios. However, the precise 'optimal' prediction is almost impossible. In this work two TRI-preconditioners were produced. The first, TRIx assumes a row-wise numbering and is optimal when refinement is made in the x-direction. The TRIy-preconditioner is similarly defined, but for (virtually) columnwise numbering. For refinements in both directions, both were employed in an alternating ADI-strategy. The results from the TRIx or TRIy and the ADI methods differ on isotropic grids even, such that ADI should be preferred in many cases. We start with the cartesian case.

MG(TRIx), Dirichlet, Cartesian, smooth=2				MG(ADI)	
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.52-1 (7)	.13+0 (10)	.23+0 (14)	.87-1 (8)	.93-1 (8)
6	.50-1 (7)	.13+0 (10)	.23+0 (14)	.11+0 (10)	.11+0 (10)
7	.48-1 (7)	.13+0 (11)	.22+0 (14)	.20+0 (13)	.28+0 (17)
8	.45-1 (7)	.13+0 (11)	.22+0 (15)	.26+0 (16)	.40+0 (24)

The next table shows that cylindrical coordinates lead to even slightly better convergence rates. Surprisingly, the results are better on moderately anisotropic meshes due to the aspect ratios by the matrix coefficients. In addition to JAC and GS calculations, TRI even works well for spherical coordinates and improves for large aspect ratios. In contrast to TRIy, TRIx does much worse due to the matrix coefficients.

MG(TRIx), Dirichlet, Cylinder, smooth=2				MG(ADI)	
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.15+0 (11)	.13+0 (10)	.22+0 (14)	.12+0 (10)	.15+0 (11)
6	.18+0 (13)	.13+0 (11)	.22+0 (14)	.14+0 (11)	.15+0 (11)
7	.21+0 (14)	.13+0 (11)	.22+0 (15)	.14+0 (11)	.15+0 (11)
8	.23+0 (16)	.13+0 (11)	.22+0 (15)	.13+0 (11)	.13+0 (11)

For spherical coordinates, the TRIy is used. TRIx is much worse.

MG(TRIy), Dirichlet, Spherical, smooth=2				MG(ADI)	
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.11+0 (10)	.73-1 (10)	.87-1 (9)	.10+0 (9)	.99-1 (9)
6	.13+0 (12)	.89-1 (11)	.87-1 (10)	.12+0 (10)	.11+0 (10)
7	.15+0 (13)	.11+0 (11)	.92-1 (11)	.12+0 (11)	.12+0 (11)
8	.17+0 (15)	.13+0 (12)	.12+0 (11)	.12+0 (11)	.12+0 (11)

All results were calculated to gain 8 digits, except of level 8/(.25,.25,.5) which was set to gain 6 digits only. Next, we apply Neumann boundary conditions on the left and bottom, which leads to very similar results.

MG(TRIx), Neumann, Cartesian, smooth=2				MG(ADI)	
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.84-1 (8)	.13+0 (10)	.23+0 (14)	.13+0 (10)	.17+0 (11)
6	.87-1 (9)	.13+0 (10)	.23+0 (14)	.17+0 (12)	.19+0 (13)
7	.86-1 (9)	.13+0 (11)	.23+0 (14)	.21+0 (14)	.25+0 (15)
8	.86-1 (9)	.13+0 (11)	.20+0 (11)	.23+0 (15)	.26+0 (13)

MG(TRIx), Neumann, Cylinder, smooth=2				MG(ADI)	
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.46-1 (7)	.12+0 (10)	.22+0 (14)	.13+0 (10)	.21+0 (13)
6	.46-1 (7)	.12+0 (11)	.22+0 (14)	.16+0 (12)	.18+0 (12)
7	.44-1 (7)	.13+0 (11)	.22+0 (15)	.18+0 (13)	.19+0 (13)
8	.42-1 (7)	.13+0 (11)	.22+0 (15)	.18+0 (13)	.16+0 (9)

MG(TRIy), Neumann, Spherical, smooth=2				MG(ADI)	
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	.95-1 (9)	.63-1 (8)	.76-1 (8)	.10+0 (10)	.11+0 (10)
6	.11+0 (10)	.58-1 (7)	.70-1 (8)	.93-1 (8)	.13+0 (11)
7	.14+0 (11)	.10+0 (10)	.62-1 (8)	.19+0 (13)	.13+0 (11)
8	.24+0 (13)	.49-1 (7)	.23-1 (6)	.12+0 (11)	.13+0 (11)

The values on level 8, especially 0.26 for (.25,.25,.5) above, are 'optimal' and could not be improved!

Surprisingly, the ADI out-did the TRI preconditioners, in some cases even on isotropic meshes! This is the result of the following table.

MG(ADI), Dirichlet, Cartesian, smooth=2			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
5	.32-1	.61-1	.79-1
6	.31-1	.76-1	.91-1
7	.30-1	.86-1	.99-1
8	.30-1	.10+0	.89-1

MG(ADI), Neumann, Cartesian, smooth=2			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
5	.46-1	.10+0	.15+0
6	.46-1	.13+0	.18+0
7	.45-1	.16+0	.17+0
8	.45-1	.18+0	.95+0

MG(ADI), Dirichlet, Cylinder, smooth=2			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
5	.12+0	.88-1	.92-1
6	.15+0	.91-1	.99-1
7	.18+0	.89-1	.91-1
8	.21+0	.86-1	.65-1

MG(ADI), Neumann, Cylinder, smooth=2			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
5	.28-1	.50-1	.67-1
6	.28-1	.52-1	.64-1
7	.27-1	.55-1	.63-1
8	.27-1	.52-1	.62-1

MG(ADI), Dirichlet, Spherical, smooth=2			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
5	.12+0	.95-1	.14+0
6	.15+0	.11+0	.15+0
7	.18+0	.14+0	.17+0
8	.20+0	.16+0	.18+0

MG(ADI), Neumann, Spherical, smooth=2			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
5	.17+0	.10+0	.29+0
6	.23+0	.92-1	.13+0
7	.29+0	.19+0	.12+0
8	.45+0	.14+0	.11+0

Next, isotropic refinement in the x-direction is used and xmax is adjusted so that the aspect-ratio is globally constant. All convergence results are 'optimal' w.r.t. the chosen damping parameters ω, even if they show sudden convergence problems on the finest level only.

AR→	1	2	3	4	5	10	100	1000
Level								
MG(TRIx), Dirichlet, Cartesian, smooth=2								
5	.52-1	.28-1	.29-1	.26-1	.23-1	.18-1	.25-1	.30-1
6	.50-1	.23-1	.25-1	.25-1	.25-1	.17-1	.21-1	.25-1
7	.48-1	.19-1	.20-1	.20-1	.20-1	.18-1	.17-1	.21-1
MG(TRIx), Dirichlet, Cylinder, smooth=2								
5	.15+0	.35-1	.33-1	.32-1	.28-1	.21-1	.32-1	.39-1
6	.18+0	.40-1	.33-1	.31-1	.31-1	.22-1	.25-1	.35-1
7	.21+0	.35-1	.32-1	.29-1	.27-1	.28-1	.24-1	.31-1
MG(TRIy), Dirichlet, Spherical, smooth=2								
5	.11+0	.43-1	.45-1	.46-1	.45-1	.44-1	.43-1	.18+0
6	.13+0	.54-1	.47-1	.55-1	.55-1	.59-1	.53-1	.32+0
7	.15+0	.66-1	.54-1	.56-1	.56-1	.62-1	.63-1	.53+0

MG(TRIx), Neumann, Cartesian, smooth=2								
AR→	1	2	3	4	5	10	100	1000
5	.84-1	.42-1	.48-1	.39-1	.39-1	.50-1	.37-1	.41-1
6	.86-1	.30-1	.41-1	.38-1	.36-1	.44-1	.33-1	.37-1
7	.86-1	.25-1	.35-1	.36-1	.33-1	.33-1	.29-1	.33-1
MG(TRIx), Neumann, Cylinder, smooth=2								
5	.45-1	.34-1	.36-1	.35-1	.42-1	.53-1	.43-1	.48-1
6	.45-1	.23-1	.37-1	.40-1	.39-1	.47-1	.38-1	.44-1
7	.43-1	.19-1	.33-1	.39-1	.37-1	.42-1	.35-1	.38-1
MG(TRIy), Neumann, Spherical, smooth=2								
5	.95-1	.42-1	.43-1	.41-1	.39-1	.36-1	.58-1	.50-1
6	.12+0	.50-1	.46-1	.44-1	.42-1	.43-1	.68-1	.60-1
7	.14+0	.83-1	.80-1	.78-1	.76-1	.75-1	.63-1	.15+0

MG(ADI), Dirichlet, Cartesian, smooth=2								
AR→	1	2	3	4	5	10	100	1000
5	.32-1	.32-1	.30-1	.35-1	.35-1	.57-1	.12+0	.11+0
6	.31-1	.29-1	.33-1	.36-1	.37-1	.69-1	.11+0	.10+0
7	.30-1	.26-1	.24-1	.32-1	.34-1	.62-1	.90-1	.10+0

MG(ADI), Dirichlet, Cylinder, smooth=2								
5	.12+0	.46-1	.40-1	.42-1	.41-1	.69-1	.14+0	.17+0
6	.15+0	.58-1	.45-1	.43-1	.42-1	.85-1	.12+0	.16+0
7	.18+0	.69-1	.44-1	.42-1	.40-1	.68-1	.11+0	.14+0

MG(ADI), Dirichlet, Spherical, smooth=2								
5	.12+0	.58-1	.48-1	.47-1	.47-1	.51-1	.46-1	.60-1
6	.15+0	.71-1	.59-1	.55-1	.55-1	.61-1	.56-1	.88-1
7	.18+0	.73-1	.64-1	.58-1	.56-1	.63-1	.78-1	.11+0

MG(ADI), Neumann, Cartesian, smooth=2								
AR→	1	2	3	4	5	10	100	1000
5	.46-1	.30-1	.38-1	.43-1	.46-1	.82-1	.13+0	.12+0
6	.46-1	.28-1	.34-1	.39-1	.42-1	.93-1	.12+0	.11+0
7	.45-1	.26-1	.30-1	.35-1	.38-1	.84-1	.11+0	.12+0

MG(ADI), Neumann, Cylinder, smooth=2								
5	.28-1	.27-1	.33-1	.36-1	.44-1	.76-1	.15+0	.17+0
6	.28-1	.28-1	.31-1	.40-1	.43-1	.96-1	.14+0	.16+0
7	.27-1	.26-1	.28-1	.37-1	.40-1	.89-1	.13+0	.16+0

MG(ADI), Neumann, Spherical, smooth=2								
5	.17+0	.98-1	.91-1	.88-1	.85-1	.86-1	.93-1	.59-1
6	.23+0	.12+0	.11+0	.10+0	.11+0	.10+0	.97-1	.43-1
7	.29+0	.16+0	.15+0	.15+0	.15+0	.11+0	.79-1	.35-1

We have the following conclusions for TRI, respective ADI-based smoothers.

- **aspect ratios:** TRI/ADI improves for large aspect ratios, but gets worse for isotropic.
- **boundary conditions:** TRI/ADI works for all; best combinations are cartesian/Dirichlet or cylindrical/Neumann.
- **coordinate systems:** Similar convergence behavior, but cylindrical

or spherical coordinates may be even better through more anisotropies which result in more tridiagonal matrices.

Choose TRI/ADI for meshes with very large aspect ratios; may not be optimal for isotropic meshes.

3.1.4 Tests on the ILU preconditioner

Tests show that the standard ILU preconditioner works best with ω-values between 0.8 and 1.0, but values $\omega \leq 1.0$ seem to be absolutely necessary in some cases. However, it is not clear how to detect the optimal damping parameters via a priori strategies. We perform the same tests as before:

MG(ILU), Dirichlet, Cartesian, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	0.38-2 (4)	0.30-2 (4)	0.25-2 (4)	0.39-2 (4)	0.46-2 (4)
6	0.38-2 (4)	0.38-2 (4)	0.53-2 (4)	0.57-2 (4)	0.61-2 (4)
7	0.37-2 (4)	0.42-2 (4)	0.87-2 (5)	0.82-2 (5)	0.15-1 (5)
8	0.34-2 (4)	0.65-2 (5)	0.35-1 (7)	0.92-2 (5)	0.26-1 (6)

MG(ILU), Dirichlet, Cylinder, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	0.33-1 (6)	0.40-1 (7)	0.32-1 (6)	0.33-1 (6)	0.19-1 (5)
6	0.51-1 (7)	0.52-1 (7)	0.37-1 (7)	0.46-1 (7)	0.30-1 (6)
7	0.69-1 (9)	0.60-1 (8)	0.37-1 (7)	0.53-1 (8)	0.34-1 (7)
8	0.84-1 (9)	0.62-1 (8)	0.22-1 (6)	0.56-1 (8)	0.36-1 (7)

MG(ILU), Spherical, Dirichlet, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	0.20-1 (5)	0.95-2 (5)	0.52-2 (4)	0.88-2 (5)	0.45-2 (4)
6	0.36-1 (7)	0.22-1 (5)	0.19-1 (5)	0.16-1 (5)	0.17-1 (5)
7	0.51-1 (7)	0.35-1 (6)	0.28-1 (6)	0.21-1 (5)	0.16-1 (5)
8	0.65-1 (8)	0.48-1 (6)	0.39-1 (7)	0.25-1 (6)	0.78-2 (5)

As can be seen, the results are very excellent. Next, we perform the tests with Neumann boundary conditions at the left and bottom parts.

MG(ILU), Neumann, Cartesian, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	0.12-1 (5)	0.61-2 (4)	0.12-1 (5)	0.12-1 (5)	0.28-1 (6)
6	0.12-1 (5)	0.61-2 (4)	0.13-1 (5)	0.13-1 (5)	0.37-1 (6)
7	0.12-1 (5)	0.93-2 (5)	0.30-1 (6)	0.16-1 (5)	0.53-1 (7)
8	0.12-1 (5)	0.95-2 (5)	div	0.211-1 (6)	div

MG(ILU), Neumann, Cylinder, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	0.30-2 (4)	0.39-2 (4)	0.10-1 (5)	0.46-2 (4)	0.13-1 (5)
6	0.33-2 (4)	0.32-2 (4)	0.86-2 (5)	0.50-2 (4)	0.96-2 (5)
7	0.33-2 (4)	0.35-2 (4)	0.86-2 (5)	0.68-2 (5)	0.12-1 (5)
8	0.31-2 (4)	0.41-2 (4)	0.10-1 (5)	0.80-2 (5)	div

MG(ILU), Spherical, Neumann(lb), smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	0.21-2 (4)	0.19-2 (4)	0.37-2 (4)	0.19-2 (4)	0.34-2 (4)
6	0.26-2 (4)	0.27-2 (4)	0.13-1 (5)	0.27-2 (4)	0.13-1 (5)
7	0.26-2 (4)	0.28-2 (4)	0.11-1 (5)	0.27-2 (5)	0.11-1 (5)
8	0.24-2 (4)	0.27-2 (4)	0.72-2 (5)	0.26-2 (5)	0.69-2 (5)

The ILU preconditioner method works very well with high aspect-ratios in one or two directions. However, there were some cases when ILU did not converge at all, examining lots of damping (ω) values. So standard ILU does not seem to be always robust! Additionally, although the numbers in these tables are very good, one must also consider efficiency and CPU timings.

For example, in three dimensions, a 27-star matrix would require a lot of time to factorize; one time for each level. And much more critical are the additional storage costs for holding the preconditioning matrix.

We derive the following conclusions ILU smoothing:

- **aspect ratios:** Excellent for (almost) all aspect ratios.
- **boundary conditions:** Works for all.
- **coordinate systems:** Excellent, but there are some problems for very high aspect ratios on highly refined meshes, especially with Neumann

boundary conditions.

Excellent multigrid rates, but most storage and CPU cost. Is there an alternative?

3.1.5 Tests on the GSTRI and GSADI preconditioners

Tests show that GSTRI and GSADI smoothers work best with ω-values always equal to 1; that means from a priori studies, the choice of $\omega = 1$ is always a very good setting. The convergence behavior of these smoothers are very good, they are probably the best choice for performing PDEs since they perform as well as ILU but in contrast they have timing rates far better than ILU and the storage cost are much less. We perform the same tests as before.

MG(GSTRI), Dirichlet, Cartesian, smooth=2			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
5	0.17-1	0.31-1	0.55-1
6	0.17-1	0.31-1	0.53-1
7	0.16-1	0.30-1	0.51-1
8	0.16-1	0.30-1	0.52-1

MG(GSTRI), Dirichlet, Cylinder, smooth=2			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
5	0.84-1	0.50-1	0.52-1
6	0.10+0	0.54-1	0.49-1
7	0.13+0	0.56-1	0.47-1
8	0.15+0	0.58-1	0.47-1

As with the TRI method, GSTRIy is employed for spherical coordinates.

MG(GSTRIy), Dirichlet, Spherical, smooth=2			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
5	0.54-1	0.33-1	0.27-1
6	0.76-1	0.51-1	0.46-1
7	0.96-1	0.67-1	0.56-1
8	0.11+0	0.83-1	0.70-1

Under Cartesian/Dirichlet boundary conditions, the GSTRI preconditioner tends to work best with 'isotropic' meshes, whereas when cylindrical or spherical coordinates are used, the preconditioner behaves better on non-isotropic meshes.

MG(GSTRI), Neumann, Cartesian, smooth=2			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
5	0.42-1	0.32-1	0.55-1
6	0.42-1	0.31-1	0.53-1
7	0.41-1	0.30-1	0.72-1
8	0.41-1	0.32-1	div

MG(GSTRI), Neumann, Cylinder, smooth=2			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
5	0.16-1	0.26-1	0.42-1
6	0.16-1	0.27-1	0.45-1
7	0.15-1	0.26-1	0.46-1
8	0.15-1	0.28-1	0.48-1

MG(GSTRIy), Neumann, Spherical, smooth=2			
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)
5	0.17-1	0.16-1	0.22-1
6	0.21-1	0.17-1	0.34-1
7	0.41-1	0.17-1	0.24-1
8	0.85-1	0.16-1	0.58-1

When Neumann boundary conditions are implemented, the preconditioner does well on almost all meshes, but moderate aspect ratios are preferred, when using cartesian or spherical coordinates, whereas 'isotropic' meshes are preferred with cylinder coordinates.

The GSADI preconditioner behaves similar to the GSTRI preconditioner, but has even better convergence rates. Higher aspect ratios are preferred with cylindrical or spherical coordinates and 'isotropic' aspect ratios when

using cartesian coordinates. When employing cylindrical coordinates, the preconditioner converged worse for isotropic meshes.

MG(GSADI), Dirichlet, Cartesian, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	0.17-1	0.25-1	0.34-1	0.30-1	0.33-1
6	0.16-1	0.27-1	0.32-1	0.30-1	0.32-1
7	0.16-1	0.29-1	0.31-1	0.35-1	0.43-1
8	0.15-1	0.28-1	0.33-1	0.36-1	0.47-1

MG(GSADI), Dirichlet, Cylinder, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	0.92-1	0.61-1	0.45-1	0.55-1	0.61-1
6	0.11+0	0.66-1	0.48-1	0.62-1	0.50-1
7	0.14+0	0.68-1	0.45-1	0.63-1	0.48-1
8	0.16+0	0.71-1	0.34-1	0.64-1	0.36-1

MG(GSADI), Dirichlet, Spherical, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	0.59-1	0.46-1	0.41-1	0.38-1	0.44-1
6	0.80-1	0.62-1	0.58-1	0.43-1	0.47-1
7	0.99-1	0.79-1	0.70-1	0.48-1	0.43-1
8	0.11+0	0.95-1	0.84-1	0.50-1	0.36-1

For Neumann boundary conditions, the preconditioner preferred cartesian and spherical coordinates for moderate anisotropies, whereas cylindrical situations preferred 'isotropic' meshes.

MG(GSADI), Neumann, Cartesian, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	0.38-1	0.32-1	0.37-1	0.54-1	0.56-1
6	0.36-1	0.32-1	0.44-1	0.47-1	0.63-1
7	0.36-1	0.42-1	0.52-1	0.52-1	0.75-1
8	0.36-1	0.52-1	div	0.61-1	div

MG(GSADI), Neumann, Cylinder, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	0.17-1	0.27-1	0.34-1	0.48-1	0.61-1
6	0.17-1	0.27-1	0.37-1	0.52-1	0.64-1
7	0.16-1	0.26-1	0.36-1	0.56-1	0.68-1
8	0.15-1	0.25-1	0.34-1	0.57-1	div

MG(GSADI), Neumann, Spherical, smooth=2					
Level	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
5	0.25+0	0.71-1	0.49-1	0.41-1	0.33-1
6	0.55+0	0.10+0	0.55-1	0.60-1	0.37-1
7	div	0.83-1	0.58-1	0.54-1	0.42-1
8	div	0.75-1	0.59-1	0.47-1	0.48-1

It is somewhat strange that GSADI has convergence problems only in the isotropic case for spherical coordinates.

Next, isotropic refinement in the x-direction is used and xmax is adjusted so that the aspect-ratio is globally constant.

AR→	1	2	3	4	5	10	100	1000
Level								
MG(GSTRIx), Dirichlet, Cartesian, smooth=2								
5	.17-1	.50-2	.32-2	.22-2	.16-2	.18-2	.20-2	.28-2
6	.17-1	.43-2	.30-2	.23-2	.19-2	.15-2	.17-2	.23-2
7	.16-1	.34-2	.25-2	.20-2	.17-2	.15-2	.19-2	.19-2
MG(GSTRIx), Dirichlet, Cylinder, smooth=2								
5	.84-1	.22-1	.14-1	.84-2	.60-2	.16-2	.33-2	.40-2
6	.10+0	.25-1	.17-1	.13-1	.10-1	.33-2	.21-2	.35-2
7	.13+0	.31-1	.18-1	.14-1	.12-1	.49-2	.17-2	.24-2
MG(GSTRIx), Dirichlet, Spherical, smooth=2								
5	div	div	div	div	div	div	div	.27-1
6	div	div	div	div	div	div	div	.26-1
7	div	div	div	div	div	div	div	.32-1
MG(GSTRIy), Dirichlet, Spherical, smooth=2								
5	.54-1	.22-1	.20-1	.18-1	.17-1	.18-1	.12-1	.65-1
6	.76-1	.30-1	.27-1	.25-1	.24-1	.20-1	.17-1	.16+0
7	.96-1	.32-1	.29-1	.27-1	.25-1	.27-1	.23-1	.31+0

MG(GSTRIx), Neumann, Cartesian, smooth=2								
AR→	1	2	3	4	5	10	100	1000
5	.42-1	.84-2	.47-2	.36-2	.28-2	.18-2	.20-2	.28-2
6	.42-1	.69-2	.37-2	.29-2	.25-2	.16-2	.17-2	.23-2
7	.41-1	.42-2	.28-2	.22-2	.19-2	.15-2	.19-2	.19-3
MG(GSTRIx), Neumann, Cylinder, smooth=2								
5	.16-1	.43-2	.28-2	.20-2	.15-2	.16-2	.33-2	.40-2
6	.16-1	.35-2	.25-2	.19-2	.16-2	.14-2	.21-2	.35-2
7	.15-1	.27-2	.20-2	.16-2	.14-2	.16-2	.17-2	.24-2
MG(GSTRIy), Neumann, Spherical, smooth=2								
5	.17-1	.14-1	.13-1	.12-1	.12-1	.10-1	.83-2	.11-1
6	.21-1	.15-1	.14-1	.13-1	.13-1	.12-1	.10-1	.96-2
7	.41-1	.16-1	.15-1	.14-1	.13-1	.13-1	.99-2	.16-1

MG(GSADI), Dirichlet, Cartesian, smooth=2								
AR→	1	2	3	4	5	10	100	1000
5	.17-1	.11-1	.14-1	.20-1	.20-1	.10-1	.26-1	.32-1
6	.16-1	.93-1	.12-1	.18-1	.18-1	.15-1	.22-1	.28-1
7	.16-1	.76-2	.92-2	.10-1	.15-1	.15-1	.19-1	.25-1
MG(GSADI), Dirichlet, Cylinder, smooth=2								
5	.92-1	.34-1	.21-1	.18-1	.18-1	.13-1	.32-1	.41-1
6	.11+0	.38-1	.27-1	.18-1	.16-1	.13-1	.29-1	.38-1
7	.14+0	.45-1	.28-1	.22-1	.15-1	.17-1	.26-1	.35-1
MG(GSADI), Dirichlet, Spherical, smooth=2								
5	.59-1	.24-1	.21-1	.19-1	.22-1	.19-1	.16-1	.24-1
6	.80-1	.32-1	.28-1	.26-1	.25-1	.22-1	.18-1	.35-1
7	.99-1	.34-1	.30-1	.28-1	.32-1	.28-1	.24-1	.45-1

MG(GSADI), Neumann, Cartesian, smooth=2							
AR→	1	2	4	5	10	100	1000
5	.38-1	.16-1	.24-1	.24-1	.20-1	.26-1	.32-1
6	.36-1	.13-1	.19-1	.20-1	.19-1	.22-1	.28-1
7	.36-1	.10-1	.15-1	.16-1	.16-1	.19-1	.25-1
MG(GSADI), Neumann, Cylinder, smooth=2							
5	.17-1	.10-1	.18-1	.18-1	.13-1	.32-1	.41-1
6	.17-1	.84-2	.16-1	.16-1	.19-1	.29-1	.38-1
7	.16-1	.70-2	.13-1	.14-1	.17-1	.26-1	.35-1
MG(GSADI), Neumann, Spherical, smooth=2							
5	.25+0	.15+0	.14+0	.14+0	.13+0	.19-1	.27-1
6	.55+0	.42+0	.41+0	.40+0	.40+0	.16-1	.22-1
7	(>150)	(>150)	(>150)	(>150)	(>150)	.15-1	.16-1

Consequence for the GSTRI/GSADI smoothers:

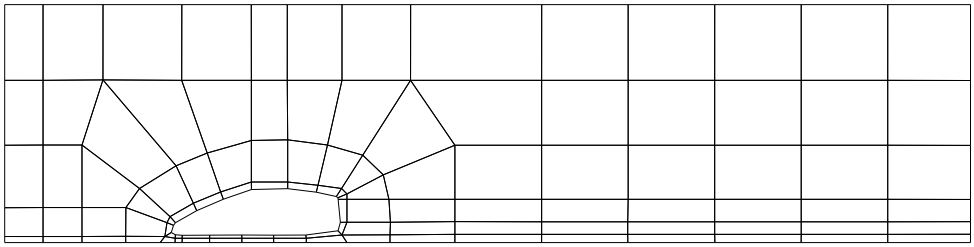
- **aspect ratios:** The GSTRI variants seem to work almost independent of the aspect ratio; in fact, only for regular meshes and spherical coordinates (GSADI), resp. for globally large aspect ratio and spherical coordinates (GSTRI), there seem to be problems.
- **boundary conditions:** Works almost perfectly for both variants.

- **coordinate systems:** Only some problems with cartesian configurations for spherical coordinates.

Favorite choice for 'Black-Box' solver, since convergence rates, CPU timings and storage cost are attractive and the damping parameter ω is fixed.

3.2 Evaluation on arbitrary quadrilaterals

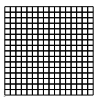
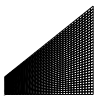
In practicable applications, the shape of the macros, that means convex quadrilaterals in our case, can be almost arbitrary. Typically, a complex coarse mesh is prescribed and the single elements are our macros. For a better motivation of the different macro types, we show again the (typical) mesh for a 'flow around the car' problem. See also the discussion of the resulting macros in section 1.2.



It is clear, that the behavior of multigrid convergence is not only determined by the local refinement made within each element, but also the shape of the element is important. The next tables show the convergence rates of various smoothers for 'isotropicly' refined odd shaped macro elements. The tables were done using two smoothing steps, 'optimum' damping parameter ω, and Dirichlet boundary conditions on level 6, i.e. h=$2^{-6} \cdot H_{macro}$.

In the following table, we give both definitions for aspect ratio AR; hereby corresponds the first value to the classical defined through the midpoint-oriented local coordinate system (see Section 1.2). As can be seen, Jacobi and Gauß-Seidel work well only for the 'fully isotropic' case while the others work for both variations. Additionally, the examples demonstrate the need

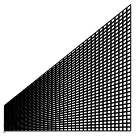
for defining the second type of aspect ratio since both definitions can vary significantly.

$AR_{old,new}$	shape	JAC	GS	TRI	ADI	GSADI	GSTRI	ILU
1.0,1.0		.44-1	.23-1	.51-1	.32-1	.17-1	.18-1	.39-2
1.8,5.0		.79+0	.49+0	.30-1	.71-1	.11-1	.85-2	.76-3
1.7,5.0		.75+0	.41+0	.28-1	.41-1	.89-2	.58-2	.60-3
1.3,5.6		.77+0	.47+0	.16+0	.93-1	.25-1	.45-1	.40-2

Again, it can be seen that Jacobi and Gauß-Seidel smoothers work well for very isotropic meshes only. In contrast, we are confirmed that ILU, (GS)TRI and GSADI are very robust smoothers.

Because the ILU method has much larger computational cost, convergence rates among the GS, TRI, ADI, GSTRI and GSADI are examined. Since the GSTRI preconditioner is a combination of the GS and TRI preconditioners, one should expect that it possess the advantages of both of these. The following grid is used: the fourth point, going counterclockwise from the origin, converges to the origin which leads to very anisotropic meshes since in the limit case a triangle is approximated.

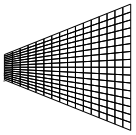
As seen in the next table, the GSTRIy preconditioner outdid the GS preconditioner even with fewer smoothing steps, especially on the more anisotropic meshes.



Cartesian, smooth=2

	$y_4 \rightarrow$	1.0	0.5	0.25	0.125	0.0625
GS	5	.23-1	.37-1	.30+0	.50+0	.77+0
smooth=2	6	.22-1	.42-1	.37+0	.68+0	.83+0
	7	.21-1	.39-1	.40+0	.74+0	.88+0
GS	5	.44-2	.34-2	.14-1	.10+0	.37+0
smooth=8	6	.44-2	.35-2	.25-1	.22+0	.50+0
	7	.42-2	.30-2	.27-1	.31+0	.61+0
TRIy	5	.52-1	.33-1	.35-1	.44-1	.47-1
	6	.50-1	.31-1	.29-1	.30-1	.41-1
	7	.48-1	.31-1	.25-1	.25-1	.26-1
GSTRIy	5	.17-1	.10-1	.11-1	.11-1	.11-1
	6	.17-1	.83-2	.84-2	.85-2	.86-2
	7	.16-1	.49-2	.43-2	.41-2	.40-2

In the next table both y-values on the left side converge symmetrically to 0.5.

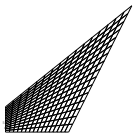


Cartesian, smooth=2

	$y_1 \rightarrow$	1.0	0.25	0.375	0.4375	0.46875
GS	5	.23-1	.32-1	.25+0	.55+0	.70+0
smooth=2	6	.22-1	.29-1	.27+0	.64+0	.83+0
	7	.21-1	.26-1	.28+0	.67+0	.87+0
GS	5	.44-2	.32-2	.11-1	.10+0	.28+0
smooth=8	6	.44-2	.30-2	.12-1	.17+0	.47+0
	7	.42-2	.24-2	.11-1	.21+0	.57+0
TRIy	5	.52-1	.34-1	.30-1	.33-1	.45-1
	6	.50-1	.36-1	.28-1	.29-1	.40-1
	7	.48-1	.36-1	.27-1	.26-1	.27-1
GSTRIy	5	.21-1	.98-2	.88-2	.83-2	.66-2
	6	.17-1	.79-2	.58-2	.53-2	.51-2
	7	.17-1	.51-2	.43-2	.39-2	.37-2

The results of the above tables show for the near square situations that GS converged better, while in the other cases, the TRI did better. However, the GSTRI as combination of both schemes, is clearly preferable.

In the final tests, the two neighbored vertices of the origin are adjusted, while observing convergence. Here, a=upper-left point and b=bottom-right point. The table shows that GSTRI could handle extreme shapes better than TRI, but this is true for 'isotropicly'-refined elements only. In some situations, the tridiagonal smoothers could handle odd shape much better than ILU.



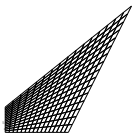
Cartesian, smooth=2, level 7

a=↓,b=→	Dirichlet				Neumann			
TRIx	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.17	.15	.15	.14	.15	.13	.12	.11
0.125	.84	.83	.82	.82	div	div	div	div
TRIy	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.17	.58	.84	.93	.11	.50	.80	.92
0.125	.15	.55	.82	.93	.12	.44	.76	.91
GSTRIx	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.35-1	.35-1	.38-1	.39-1	.10	.87-1	.81-1	.78-1
0.125	.69	.68	.67	.66	div	div	.70	.61
GSTRIy	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.34-1	.33	.69	.86	.35-1	.29	.64	.84
0.125	.38-1	.30	.67	.85	.53-1	.24	.60	.81
ADI	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.43-1	.85-1	.10	.10	.82-1	.10	.12	.13
0.125	.10	.15	.19	.25	.16	.17	.21	.25
GSADI	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.18-1	.25-1	.37-1	.39-1	.56-1	.51-1	.43-1	.42-1
0.125	.21-1	.40-1	.73-1	.76-1	.43-1	.54-1	.73-1	.76-1
ILU	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.37-2	.23-1	.56	div	.10-1	.25-1	div	div
0.125	.34-2	.54-1	div	div	.16-1	.11	div	div

'div' stands for such cases that even small-scaled variations of the damping

parameter ω did not lead to convergence in less than 300 multigrid steps. The result is that only ADI and GSADI lead to excellent convergence in all cases while GSADI is even preferable, due to better convergence rates and particularly since ω=1 is always the optimal choice.

Next, tests are made using cylindrical coordinates. Again, the standard ILU smoother has some problems



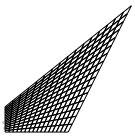
Cylinder, smooth=2, level 7

a=↓,b=→	Dirichlet				Neumann			
TRIx	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.28	.25	.23	.23	.95-1	.97-1	.10	.12
0.125	.86	.85	.84	.83	.76	.73	.70	.68
TRIy	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.17	.57	.83	.93	.10	.47	.76	.90
0.125	.15	.54	.81	.92	.10	.45	.74	.88
GSTRIx	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.12+0	.98-1	.91-1	.79-1	.22-1	.37-1	.52-1	.54-1
0.125	.73	.71	.69	.67	.57	.53	.50	.47
GSTRIy	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.39-1	.32	.68	.85	.23-1	.26	.59	.80
0.125	.52-1	.29	.64	.83	.52-1	.23	.56	.77
ADI	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.11	.10	.12	.14	.39-1	.10	.12	.14
0.125	.12	.19	.22	.26	.12	.19	.22	.26
GSADI	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.63-1	.57-1	.54-1	.55-1	.19-1	.36-1	.51-1	.54-1
0.125	.23-1	.71-1	.93-1	.11	.23-1	.71-1	.93-1	.11
ILU	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.18-1	.21-1	.54	div	.31-2	.18-1	div	div
0.125	.35-2	.68-1	div	div	.60-2	.69-1	div	div

As before, ADI and particularly GSADI are the winners.

Finally, tests are made using spherical coordinates. Most smoothers have

difficulties in these coordinates. For Dirichlet boundary conditions, only the TRIy, ADI, GSTRIy and GSADI smoother converged. For Neumann boundary conditions, almost all smoothers had problems which for ADI and GSADI could be solved by more smoothing steps.



Spherical, smooth=2, level 7

a=↓,b=→	Dirichlet				Neumann			
TRIx	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	div	div	div	div	div	div	div	div
0.125	div	div	div	div	div	div	div	div
TRIy	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.12	.54	.82	.93	.86	.90	div	div
0.125	.11	.54	.79	.90	div	div	div	div
GSTRIy	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.55-1	.34	.68	div	.12	.32	.62	.77
0.125	.52-1	.35	.64	div	.48	.33	.58	.69
ADI	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.97-1	.35	.56	.59	div	div	div	div
0.125	.13	.47	.62	.51	div	div	div	div
GSADI	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.38-1	.19	.37	.39	div	div	div	div
0.125	.33-1	.30	.44	.31	div	div	div	div
ILU	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	div	div	div	div	div	div	div	div
0.125	div	div	div	div	div	div	div	div

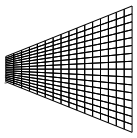
Spherical, smooth=8, level 5								
a=↓,b=→	Dirichlet				Neumann			
ADI	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.24-1	.90-1	.13	.13	.20-1	.25	.40	.30
0.125	.36-1	.15	.15	.12	..79-1	.55	.73	.41
GSADI	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.12-1	.43-1	.65-1	.59-1	.38-2	.43-1	.15	.13
0.125	.18-1	.77-1	.75-1	.52-1	.12-1	.80-1	.22	.12
ILU	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.14-2	.11	.54-1	.54-1	.49-3	div	div	div
0.125	div	div	div	div	div	div	div	div

Spherical, smooth=8, level 6								
a=↓,b=→	Dirichlet				Neumann			
ADI	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.31-1	.11	.20	.20	.12-1	.12	.23	.24
0.125	.23-1	.19	.24	.18	.80-1	.23	.27	.27
GSADI	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.17-1	.56-1	.12	.10	.26-2	.56-1	.12	.15
0.125	.11-1	.12	.14	.91-1	.10-1	.12	.16	.15
ILU	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	div	div	div	div	div	div	div	div
0.125	div	div	div	div	div	div	div	div

Spherical, smooth=8, level 7								
a=↓,b=→	Dirichlet				Neumann			
ADI	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.32-1	.11	.25	.29	.16-1	.14	.29	.32
0.125	.27-1	.18	.34	.28	.89-1	.22	.38	.29
GSADI	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	.18-1	.53-1	.15	.16	.16-2	.53-1	.17	.19
0.125	.14-1	.11	.21	.15	.81-2	.11	.24	.17
ILU	0.5	0.25	0.125	0.0625	0.5	0.25	0.125	0.0625
0.5	div	div	div	div	div	div	div	div
0.125	div	div	div	div	div	div	div	div

The ILU smoother did not converge for different ω parameters using up to 16 smoothing operations.

Finally an odd shape with local refinements is examined for cylindrical coordinates.



Cylinder, smooth=2, level 7

Dirichlet	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
ADI	.10	.55-1	.11	.46-1	.42-1
GSADI	.37-1	.24-1	.20-1	.48-2	.38-2
ILU	.41-2	.12-1	.45-2	.26-2	.57-5
Neumann	(.5,1,1)	(.5,.5,1)	(.25,.25,.5)	$(.5,.5,1)^2$	$(.25,.25,.5)^2$
ADI	.41-1	.62-1	.11	.11	.15
GSADI	.10-1	.13-1	.21-1	.23-1	.29-1
ILU	.58-3	.62-3	.39-2	.73-3	.25-1

- Not counting ILU, the GSADI performs best and is followed by GSTRI, and then ADI for Dirichlet boundary conditions.
- Careful study must be made when solving on odd shapes when using ILU: other methods converge often better.
- The TRIx smoother together with Neumann boundary conditions converges only with cylindrical coordinates.

3.3 Conclusion of the results

We finish the numerical and computational studies with the following results:

- The standard JAC and GS smoothers seem to work well for very isotropic meshes only. In addition, spherical coordinates are almost impossible. Also the choice of damping parameters ω is quite delicate. While cartesian coordinates with Dirichlet boundary conditions show some superconvergence results (compare with the paper of Yserentant), cylindrical coordinates behave better with Neumann boundary contions for both smoothers.

- Pure tridiagonal schemes are much better for largely anisotropic meshes, especially if being globally defined. However, the rates for isotropic meshes are worse, and the choice of ω is not clear. ADI, that means switching of rowwise and columnwise tridiagonal numbering, is preferable.

- ILU shows often the best convergence rates, but the numerical and computational costs are much larger. Additionally, for very anisotropic meshes, the choice of ω is not clear, and the standard ILU may diverge.

- Our favorite is GSTRI, resp. GSADI which combines the advantages of GS and TRI schemes: that means, uniformally excellent for isotropic and anisotropic meshes, and the choice of ω=1 is always optimal for GSADI. Since also the computational efficiency is very high (see the FEAST Indices) and the additional storage is negligible, GSADI is our actual favorite for all coordinate systems and all types of boundary conditions.

Chapter 4

Conclusions

The aim of this work was to study efficient (multigrid) Poisson solvers in general coordinates on generalized tensorproduct meshes and to help decide which method is most appropriate. It was realized using software developed by the original FEAT group. As part of the growing FEAST project, this work has brought new implementations of sparse banded matrix solvers.

The work compared different methods ranging from the classical Gaussian elimination over Krylov-space methods up to multigrid schemes. Several smoothers were tested for different coordinate systems on meshes with varying aspect ratios and different shape.

The results of this work are:

- For small problems (up to 10,000 unknowns) many classical (one-grid) methods work well. Arguments can be made for storage cost, CPU-time and also program complexity. The multigrid program requires several prepared preconditioners for all levels, grid transfer routines and a boundary update routine which need much more implementation work. Moreover, the direct solvers have no problem at all with highly anisotropic meshes.
- For larger problems, classical methods either require inverting a matrix, which would lead to memory problems as in Gaussian elimination, or have slower convergence behavior due to small grid-steps and

corresponding bad conditioning. In contrast, the multigrid program possesses convergence behavior which is (more or less) independent of the mesh size. The extra memory and algorithmic complexities are minimal in comparison to the CPU-time gains acquired.

- The tests lead to the conclusions that, for 'small' up to moderately isotropic problems, Jacobi or Gauß-Seidel methods work well enough. For more extreme refinements, the tridiagonal-based methods, TRI, ADI, GSTRI and GSADI, work better.

- The ILU method shows often the best convergence results, but has obvious CPU and storage cost disadvantages, especially for higher levels.

From the tests on arbitrary quadrilaterals, which are a better representation of real life, this paper concludes that the GSTRI/GSADI ($\sim$ linewise Gauß-Seidel schemes) methods are the favorites, not only because of convergence behavior, but also because the choice ω=1 of the inherent parameter is optimal in most cases. This is in contrast to all other schemes, even the applied ILU smoother. Altogether, this linewise Gauß-Seidel variant is our favorite w.r.t. numerical robustness and efficiency, but also regarding the computational efficiency, since this very sophisticated scheme can be preferred in the range of hundreds of MFLOPS/s on modern processors.

Bibliography

[1] H. Blum, J. Harig, S. Müller, S. Turek: *Finite Element Analysis Tools User Manual Release 1.3*, Heidelberg 1992

[2] D. Braess: *Finite Elemente*, Springer-Verlag, 1992

[3] Chr. Becker, A. Runge, S. Turek and the FEAST Group: *The FEAST Indices - Realistic evaluation of modern software components and processor technologies*, to be printed

[4] Chr. Becker: *FEAST - The realization of Finite Element Software for high-performance applications*, Thesis, to be printed

[5] Ch. Grossmann/ H.G. Roos: *Numerik partieller Differentialgleichungen* Teubner Studienbucher, 1993

[6] A. Hujeirat, R. Rannacher: *A Method for Computing Compressible, Highly Stratified Flows in Astrophysics Based on Operator Splitting* Preprint 95-45 (SFB 359), 1995

[7] C. Johnson: *Numerical solution of partial differential equations by the finite element method*, Cambridge, 1994

[8] J. Heywood, R. Rannacher, S. Turek: *Artificial boundaries and flux and pressure conditions for the incompressible Navier-Stokes equations*, Int. J. Numer. Meth. Fluids, 22,325-352, 1996

[9] S. Kilian: *A new multigrid approach for the parallel solution of elliptic PDE's*, to be printed

[10] SPARSKIT (by Y. Saad):
http://www.cs.umn.edu/Research/arpa/SPARSKIT/sparskit.html

[11] The national technology roadmap for semiconductors, 1997 edition, http://www.sematech.org/public/roadmap/index.htm

[12] Turek, S.: *Efficient Solvers for incompressible flow problems: An Algorithmic approach in view of computational aspects*, Heidelberg, LNCSE 6, Springer, 1999

[13] Turek, S.: *Some basic concepts of FEAST*, Proc. 14th GAMM Seminar 'Concepts of Numerical Software', Kiel, January 1998, NNFM, Vieweg, 1998

[14] Turek, S.: *On the realistic performance of components in iterative-solvers*, to appear in: High Performance Scientific and Engineering Computing (H.-J. Bungartz, F. Durst, Chr. Zenger, eds.), LNCSE, Springer-Verlag, 1999

[15] Yserentant, H.: *Old and new convergence proofs for multigrid methods*, Acta Numerica, 285-326, 1993

Appendix: Developed software

For completeness, we present a few examples from the FORTRAN code which is used to realize this work.

After the computational mesh, matrices and vectors have been generated with the FEAT software, the heart (`m010.f`) of the multigrid program is executed. It has been slightly modified from the original FEAT routine. Modifications had to be made to work with sparse banded matrix storage handling, and this with high computational efficiency.

For one-grid as well as for the coarse grid solver, computations were made using either conjugate-gradient (CG) or Gaussian elimination (LU factorization) (`yex3.f`).

The defect computation routine is realized with `ymax13.f`. Here different strategies for matrix-vector multiplication can be chosen based on the actual hardware. The optimal version depends often on the type and size of cache used.

Pre- and postsmoothing is realized through the routine `ysmooth.f`. The modular programming style makes it easy to choose from different matrix-vector multiplication and smoothers.

The prolongation and restriction operators are realized in just a few lines of code (`prolong.f`). The subroutines called are DAXPY-like variants from the 'Basic Linear Algebra Subroutine' (BLAS) and are optimized often at the machine level.

There are several methods of boundary handling. In this implementation, boundary points are updated after restriction and prolongation. The routine

`ymbc13.f` loops through an array of boundary indices and simply sets them to the correct values.

Several matrix-vector multipliers are used in the program. A few, `dgbmv3` and `band16` are shown. In `band16`, one can see the basic unrolling techniques that are employed: these take less CPU-time since they make fewer reference to the index variable.

The tensorproduct mesh is realized by calling the refinement subroutine (see Section 1.2) once for each direction and then calculating the values to be stored as the mesh (`refine.f`).

The smoothers are built after the main stiffness matrix has been processed for boundary conditions. The preparation of the smoothers is made in `adi.f` and `gs.f`. First the respective part of the stiffness matrix is stored into the preconditioner memory and then these are factorized for later use as smoothers.

```
************************************************************************
* FINITE ELEMENT ANALYSIS TOOLBOX  FEAT  (Release 1.3)                 *
*                                                                      *
* Authors: H. Blum, J. Harig, S. Mueller, S. Turek, J. Wallis          *
*          Institute of Applied Mathematics                            *
*          University of Heidelberg                                    *
*          D-6900 HEIDELBERG                                           *
*                                                                      *
************************************************************************
*                                                                      *
* M010     In moment nur typ 3 (DPRSM/DPOSM/DEX aufgerufen             *
*                              mit extra vector)                       *
*                                                                      *
* Purpose  Solution of a linear system  A*X = B  using                 *
*          multigrid iteration                                         *
*          Double precision version                                    *
*                                                                      *
* Subroutines/functions called  LL21 , LLC1                            *
*                                                                      *
* Version from  08/25/90                                               *
*                                                                      *
* INPUT    TYPE                                                        *
* -----    ----                                                        *
* DX       R*8    Starting address of vectors containing the           *
* DB       R*8    solution and the right hand side, DD, DR are used as *
* DD       R*8    auxiliary vectors only                               *
* DR       R*8                                                         *
* KOFFX           The actual starting address of DX on level ILEV      *
* KOFFB           is DX(1+KOFFX(ILEV)) (analogously for DB and DD)     *
* KOFFD           Total space required for all vectors is              *
*                 KNEQ(NLMIN)+...+KNEQ(NLMAX)                          *
*                 DX(1+KOFFX(NLMAX)) contains initial solution         *
*                 DB(1+KOFFB(NLMAX)) contains right hand side          *
* KNEQ     I*4    Number of equations for all levels                   *
* NLMAX    I*4    Iteration uses levels NLMIN to NLMAX,                *
* NLMIN    I*4    NLMAX  is the finest level                           *
* NIT      I*4    Maximum number of iterations                         *
*                 Iteration completed after reaching the finest level  *
* EPS      R*8    Desired precision                                    *
*                 Stop if !!DEF!! < EPS                                *
* KPRSM    I*4    Number of pre -smoothing steps for all levels        *
* KPOSM    I*4    Number of post-smoothing steps for all levels        *
* ICYCLE   I*4    <0: special cycle types (not yet implemented)        *
*                 =0: F-Cycle                                          *
*                 =1: V-Cycle                                          *
*                 =2: W-Cycle                                          *
*                 >2: Cycle of higher order                            *
* DAX      SUBR   CALL DAX(DX,DAX,NEQ,A1,A2)                           *
*                 Returns  DAX := A1*A*DX+A2*DAX                       *
* DPROL    SUBR   DPROL(DX1,DX2)                                       *
*                 Returns  DX2 := Prolongation(DX1)  to higher level   *
* DREST    SUBR   DREST(DD1,DD2)                                       *
*                 Returns  DD2 := Restriction(DD1)  to lower level     *
* DPRSM    SUBR   DPRSM(DX,DB,DD,NEQ,NPRSM)                            *
*                 Returns  DX  after  NPRSM:=KPRSM(ILEV)               *
*                 pre-smoothing steps, DD  is used as auxiliary vector *
* DPOSM    SUBR   Same as above, used for post-smoothing               *
```

```
* DEX      SUBR   DEX(DX,DB,DD,NEQ)                                    *
*                 Returns exact solution                               *
* DBC      SUBR   DBC(DX,NEQ)                                          *
*                 Copies boundary data onto components of  DX          *
* KIT0     I*4    auxiliary vectors of length  NLMAX                   *
* KIT      I*4                                                         *
*                                                                      *
*                                                                      *
* OUTPUT   TYPE                                                        *
* ------   ----                                                        *
* DX       R*4    Solution vector on DX(KOFFX(NLMAX))                  *
* ITE      I*4    Number of iterations                                 *
* IER      I*4    Error indicator                                      *
*                                                                      *
************************************************************************
C
      SUBROUTINE M010_(DX,DB,DD,DR,KOFFX,KOFFB,KOFFD,KOFFR,
     * KNEQ,NIT,ITE,EPS,
     * DAX,DPROL,DREST,DPRSM,DPOSM,DEX,DBC,KIT0,KIT,IREL)
C
      PARAMETER (NNARR=299,NNLEV=9)
      IMPLICIT DOUBLE PRECISION (A,C-H,O-U,W-Z),LOGICAL(B)
      CHARACTER SUB*6,FMT*15,CPARAM*120
      INTEGER MDATA,NFL(NNLEV,NNLEV),IRES
c      INTEGER LMATEX,LPIVOT
      DIMENSION VWORK(1),KWORK(1)
      DOUBLE PRECISION TTSM1,TTSM2,DFLS1(NNLEV),DFLS2(NNLEV),timmat
      DOUBLE PRECISION DFLA(NNLEV),DFLR(NNLEV),DFLP(NNLEV),DFLX(NNLEV)
C
      DIMENSION DX(*),DB(*),DD(*),KOFFX(*),KOFFB(*),KOFFD(*)
      DIMENSION KNEQ(*),KIT0(*),KIT(*),DR(*),KOFFR(*)
      COMMON /ERRCTL/ IER,ICHECK
      COMMON /CHAR/   SUB,FMT(3),CPARAM
      COMMON /OUTPUT/ M,MT,MKEYB,MTERM,MERR,MPROT,MSYS,MTRC,IRECL8
      COMMON /MGTRD/  KNEL(NNLEV),KNVT(NNLEV),KNMT(NNLEV),
     *                KNVEL(NNLEV),KNVBD(NNLEV)
      COMMON /MGPAR/  ILEV,NLEV,NLMIN,NLMAX,
     *                ICYCLE,KPRSM(NNLEV),KPOSM(NNLEV)
      COMMON /MGTIME/ TTMG,TTS,TTE,TTD,TTP,TTR,IMTIME
      COMMON /XYPAR/  DXYPAR(NNARR),KXYPAR(NNARR)
      COMMON /MGPAR0/ DOMPOS(NNLEV),DOMPRS(NNLEV),OMEX,EPSEX,
     *                KIPOSM(NNLEV),KIPRSM(NNLEV),IEX,NITEX,IELE
      include "jcommons.f"
      COMMON          NWORK,IWORK,IWMAX,L(NNARR),DWORK(1)
      EQUIVALENCE (DWORK(1),VWORK(1),KWORK(1))
      SAVE /ERRCTL/,/CHAR/,/OUTPUT/,/MGTRD/,/MGPAR/,/MGTIME/,/John/

C     Preconditionors available
C      EXTERNAL TRI1,TRI2,ADITRI,JACOBI,GSTRI1,ILU,GS
C     Matrix-Vector
      EXTERNAL Lax13_,dgbmv_,dgbmv3_,DPROL,DREST,DPRSM,DPOSM,DEX
C
      SUB='M010  '
      IF (ICHECK.GE.997) CALL OTRC('M010  ','08/25/90')
      IER=0
C
      MDATA=678
```

```
      OPEN (MDATA,FILE='mflop_times.dat')
C
      BREL=IREL.EQ.1
      BMSG2=M.GE.2.OR.MT.GE.2
C
      BTIME=IMTIME.GT.0
      IF (BTIME) THEN
       IF (IMTIME.EQ.1) THEN
        TTMG=0D0
        TTS=0D0
        TTE=0D0
        TTD=0D0
        TTP=0D0
        TTR=0D0
        TTSM1=0D0
        TTSM2=0D0
        DO ILEV=NLMIN,NLMAX
           DFLS1(ILEV)=0D0
           DFLS2(ILEV)=0D0
           DFLA(ILEV)=0D0
           DFLR(ILEV)=0D0
           DFLP(ILEV)=0D0
           DFLX(ILEV)=0D0
           DO JLEV=NLMIN,NLMAX
            NFL(ILEV,JLEV)=0
           ENDDO
        ENDDO
       ENDIF
       CALL ZTIME(TTMG0)
      ENDIF
C
      NIT0=MAX(ITE,0)
      ILEV=NLMAX
      IRES=0
C
C *** special case - only one level
      IF (NLMIN.EQ.NLMAX) THEN
       CALL LCP1(DB(1+KOFFB(NLMAX)),DX(1+KOFFX(NLMAX)),KNEQ(NLMAX))
       IF (BTIME) CALL ZTIME(TTE0)
       CALL DEX(DX(1+KOFFX(NLMAX)),DB(1+KOFFB(NLMAX)),
     *    DD(1+KOFFD(NLMAX)),DR(1+KOFFR(NLMAX)),KNEQ(NLMAX),DPRSM)
       IF (BTIME) THEN
        CALL ZTIME(TTE1)
        TTE=TTE+TTE1-TTE0
       ENDIF
       GOTO 99999
      ENDIF
C     =====================================
C
C *** level counts
      KIT0(NLMAX)=1
      DO 2 ILEV=NLMIN+1,NLMAX-1
      IF (ICYCLE.EQ.0) THEN
       KIT0(ILEV)=2
      ELSE
       KIT0(ILEV)=ICYCLE
      ENDIF
```

```
2     CONTINUE
C
      IF (BTIME) CALL ZTIME(TTD0)
      CALL LCP1(DB(1+KOFFB(NLMAX)),DR(1+KOFFR(NLMAX)),KNEQ(NLMAX))
      CALL DAX(DX(1+KOFFX(NLMAX)),DR(1+KOFFR(NLMAX)),KNEQ(NLMAX),
     *         -1D0,1D0,IRES)
C
      if (Inorm.eq.1)then
         CALL LL21(DR(1+KOFFR(NLMAX)),KNEQ(NLMAX),DEF)
      else
         CALL LL21_(DR(1+KOFFR(NLMAX)),KNEQ(NLMAX),DEF)
      endif

      DEFOLD=DEF
      IRES=1
C
      IF (BMSG2) THEN
       WRITE (CPARAM,'(I15,D25.16)') 0  ,DEF
       CALL OMSG(73,'M010  ')
      ENDIF
C *** FD  is considered as initial defect
C *** (after at least one pre-smoothing step)
      FD=DEF
      IF (BTIME) THEN
       CALL ZTIME(TTD1)
       TTD=TTD+TTD1-TTD0
       x=(4+2*9)*KNEQ(NLMAX)/(TTD1-TTD0)/1D6
       DFLA(NLMAX)=DFLA(NLMAX)+x
       NFL(3,NLMAX)=NFL(3,NLMAX)+1
      ENDIF
      IF (DEF.LE.EPS.AND..NOT.BREL) THEN
       ITE=0
       GOTO 1000
      ENDIF
C
C *** Start multigrid iteration
C
      DO 100 ITE=1,NIT
C
C *** initialize level counts for all levels
      DO 101 ILEV=NLMIN,NLMAX
101   KIT(ILEV)=KIT0(ILEV)
C
      ILEV=NLMAX
C
110   IF (ILEV.NE.NLMIN) THEN
C
C *** Pre-smoothing
C
        IF (KPRSM(ILEV).GT.0) THEN
        IF (BTIME) CALL ZTIME(TTS0)
      CALL ysm3(DX(1+KOFFX(ILEV)),DB(1+KOFFB(ILEV)),
     *             DD(1+KOFFD(ILEV)),DR(1+KOFFR(ILEV)),
     *             KNEQ(ILEV),KPRSM(ILEV),IRES,
     *             DOMPRS(ILEV),dgbmv_,DPRSM)

        IF (BTIME) THEN
```

```
         CALL ZTIME(TTS1)
         TTS=TTS+TTS1-TTS0
         x=(5+2*9)*KNEQ(ILEV)/(TTS1-TTS0)/1D6
         DFLS1(ILEV)=DFLS1(ILEV)+x
         NFL(1,ILEV)=NFL(1,ILEV)+1
c         WRITE(678,*)"PRSM",ILEV,x
        ENDIF
        ENDIF
C
        IF (BTIME) CALL ZTIME(TTD0)
        CALL LCP1(DB(1+KOFFB(ILEV)),DR(1+KOFFR(ILEV)),KNEQ(ILEV))
        CALL DAX(DX(1+KOFFX(ILEV)),DR(1+KOFFR(ILEV)),KNEQ(ILEV),
     *           -1D0,1D0,IRES)
        IRES=0
C
        IF (BTIME) THEN
         CALL ZTIME(TTD1)
         TTD=TTD+TTD1-TTD0
         x=(2+2*9)*KNEQ(ILEV )/(TTD1-TTD0)/1D6
         DFLA(ILEV)=DFLA(ILEV)+x
         NFL(3,ILEV)=NFL(3,ILEV)+1
c         WRITE(678,*)"DAX",ILEV,x
        ENDIF
C
       ILEV=ILEV-1
C *** restriction of defect
C
       IF (BTIME) CALL ZTIME(TTR0)
       CALL DREST(DR(1+KOFFR(ILEV+1)),DB(1+KOFFB(ILEV)))
C
C *** choose zero as initial vector on lower level
       CALL LCL1(DX(1+KOFFX(ILEV)),KNEQ(ILEV))
       CALL DBC(DB(1+KOFFB(ILEV)),KNEQ(ILEV))
C
       IF (BTIME) THEN
        CALL ZTIME(TTR1)
        TTR=TTR+TTR1-TTR0
        x=(2.5)*KNEQ(ILEV+1)/(TTR1-TTR0)/1D6
        DFLR(ILEV+1)=DFLR(ILEV+1)+x
        NFL(4,ILEV+1)=NFL(4,ILEV+1)+1
c       WRITE(678,*)"REST",ILEV+1,x
       ENDIF
       GOTO 110
      ENDIF
C
C Exact solution on lowest level
C
      IF (BTIME) CALL ZTIME(TTE0)
      CALL DEX(DX(1+KOFFX(NLMIN)),DB(1+KOFFB(NLMIN)),DD(1+KOFFD(NLMIN)),
     *         DR(1+KOFFR(NLMIN)),KNEQ(NLMIN),DPRSM)
C
      IF (BTIME) THEN
       CALL ZTIME(TTE1)
       TTE=TTE+TTE1-TTE0
       x=5D0*DBLE(KNEQ(NLMIN))**(3D0/2D0) /(TTE1-TTE0)/1D6
       DFLX(NLMIN)=DFLX(NLMIN)+x
       NFL(6,NLMIN)=NFL(6,NLMIN)+1
```

```
      ENDIF
C
C
130   IF (ILEV.NE.NLMAX) THEN
       ILEV=ILEV+1
C *** DPROL  returns  DD:=PROL(DX)
C
      IF (BTIME) CALL ZTIME(TTP0)
       CALL DPROL(DX(1+KOFFX(ILEV-1)),DD(1+KOFFD(ILEV)))
       CALL DBC(DD(1+KOFFD(ILEV)),KNEQ(ILEV))
       CALL LLC1(DD(1+KOFFD(ILEV)),DX(1+KOFFX(ILEV)),KNEQ(ILEV),1D0,1D0)
C
      IF (BTIME) THEN
       CALL ZTIME(TTP1)
       TTP=TTP+TTP1-TTP0
       x=(4.25)*KNEQ(ILEV )/(TTP1-TTP0)/1D6
       DFLP(ILEV)=DFLP(ILEV)+x
       NFL(5,ILEV)=NFL(5,ILEV)+1
      ENDIF
C
C *** Post-smoothing
C
      IRES=0

         write(*,*)'Post-smoothing'
      IF (KPOSM(ILEV).GT.0) THEN
      IF (BTIME) CALL ZTIME(TTS0)
        CALL ysm3(DX(1+KOFFX(ILEV)),DB(1+KOFFB(ILEV)),
     *              DD(1+KOFFD(ILEV)),DR(1+KOFFR(ILEV)),
     *              KNEQ(ILEV),KPRSM(ILEV),IRES,
     *              DOMPRS(ILEV),dgbmv_,DPOSM)
C
      IF (BTIME) THEN
       CALL ZTIME(TTS1)
       TTS=TTS+TTS1-TTS0
       x=(5+2*9)*KNEQ(ILEV)/(TTS1-TTS0)/1D6
       DFLS2(ILEV)=DFLS2(ILEV)+x
       NFL(2,ILEV)=NFL(2,ILEV)+1
c       WRITE(678,*)"POSM",ILEV,x
      ENDIF
      ENDIF
C
       KIT(ILEV)=KIT(ILEV)-1
       IF (KIT(ILEV).EQ.0) THEN
        IF (ICYCLE.EQ.0) THEN
         KIT(ILEV)=1
        ELSE
         KIT(ILEV)=KIT0(ILEV)
        ENDIF
        GOTO 130
       ELSE
        GOTO 110
       ENDIF
      ENDIF
C
      IF (BTIME) CALL ZTIME(TTD0)
      CALL LCP1(DB(1+KOFFB(NLMAX)),DR(1+KOFFR(NLMAX)),KNEQ(NLMAX))
```

```
      CALL DAX(DX(1+KOFFX(NLMAX)),DR(1+KOFFR(NLMAX)),KNEQ(NLMAX),
     *         -1D0,1D0,IRES)

      if (Inorm.eq.1)then
         CALL LL21(DR(1+KOFFR(NLMAX)),KNEQ(NLMAX),DEF)
      else
         CALL LL21_(DR(1+KOFFR(NLMAX)),KNEQ(NLMAX),DEF)
      endif
      IRES=1
      DEFOLD=DEF
C
      IF (BTIME) THEN
       CALL ZTIME(TTD1)
       TTD=TTD+TTD1-TTD0
       x=(4+2*9)*KNEQ(NLMAX )/(TTD1-TTD0)/1D6
       DFLA(NLMAX)=DFLA(NLMAX)+x
       NFL(3,NLMAX)=NFL(3,NLMAX)+1
c       WRITE(678,*)"DAX",ILEV,x
      ENDIF
C
      IF (BMSG2) THEN
       WRITE (CPARAM,'(I15,D25.16)') ITE,DEF
       CALL OMSG(73,'M010  ')
      ENDIF
      IF (BREL) THEN
       IF (DEF.LE.FD*EPS.AND.ITE.GE.NIT0) GOTO 1000
      ELSE
       IF (DEF.LE.EPS) GOTO 1000
      ENDIF
C
100   CONTINUE
C
      WRITE (CPARAM,'(I15,2D25.16)') NIT,DEF,DEF/FD
      CALL OMSG(71,'M010  ')
      CALL OMSG(72,'M010  ')
      IER=1
      GOTO 99999
C
1000  IER=0
      y=FD
      IF (FD.GE.1D-70) FD=DEF/FD
      WRITE (CPARAM,'(I15,2D25.16)') ITE,DEF,FD
      CALL OMSG(72,'M010  ')
      WRITE (CPARAM,'(D25.16)') FD**(1D0/DBLE(ITE))
      CALL OMSG(76,'M010  ')
C
99999 IF (BTIME) THEN
       CALL ZTIME(TTMG1)
       TTMG=TTMG+TTMG1-TTMG0-timmat
       DFLX(NLMIN)=DFLX(NLMIN)-timmat
c       write(*,*)"HAllo m010.f",-timmat
C --------------AUSGABE-------------------------------------------------
       DO ILEV=NLMIN+1,NLMAX
          NFL(6,ILEV)=1.
       ENDDO
       DO ILEV=1,5
          NFL(ILEV,1)=1.
```

```
       ENDDO
      WRITE (678,*)
     *'           TTMG,    TTS,            TTD,    TTP,    TTR,   TTE'
      WRITE (678,988) TTMG,TTS,    TTD,TTP,TTR,TTE
      WRITE (678,986) TTMG/TTMG*100,TTS/TTMG*100,
     * TTD/TTMG*100,TTP/TTMG*100,TTR/TTMG*100,TTE/TTMG*100
      WRITE (678,*)"-------------------------------------------------"
      WRITE (678,*)
     *"MFLOP:      LEV    PRSM    POSM     DAX    PROL    REST     DEX"
      ILEV=NLMIN
          x=ILEV
        WRITE(678,987)KNEQ(ILEV),x,0,0,0,0,0,DFLX(ILEV)/NFL(6,ILEV)
       DO ILEV=NLMIN+1,NLMAX
          x=ILEV
         WRITE(678,987)KNEQ(ILEV),x,DFLS1(ILEV)/NFL(1,ILEV),
     *   DFLS2(ILEV)/NFL(2,ILEV),
     *   DFLA(ILEV)/NFL(3,ILEV),DFLP(ILEV)/NFL(5,ILEV),
     *   DFLR(ILEV)/NFL(4,ILEV),DFLX(ILEV)/NFL(6,ILEV)
       ENDDO
      WRITE (678,*)"-------------------------------------------------"

       x=(4+2*9)*(1+ITE)*KNEQ(NLMAX)
       DO ILEV=NLMAX,NLMIN+1,-1
          x=x
     *   +(5*DBLE(NFL(1,ILEV))+5*DBLE(NFL(2,ILEV))
     *   +2.5*DBLE(NFL(4,ILEV))+4.25*DBLE(NFL(5,ILEV))
     *   +2*DBLE(NFL(3,ILEV)))*DBLE(KNEQ(ILEV))
     *   +(9*(2*DBLE(NFL(1,ILEV))+2*DBLE(NFL(2,ILEV))
     *   +2*DBLE(NFL(3,ILEV))))*DBLE(KNEQ(ILEV))
       ENDDO
        x=x+5D0*NFL(6,NLMIN)*DBLE(KNEQ(NLMIN))**(3D0/2D0)

       x=x/TTMG/1D6
       z=1D6*TTMG/ITE/KNEQ(NLMAX)*(-1D0)/LOG10(FD**(1D0/DBLE(ITE)))

       write(678,*) "Total MFL RATE, Efficiency"
       write(678,984) x,z

       write(678,*) "CONVERGENCE RATE, ITE, RES, init.RES"
       write(678,985) FD**(1D0/DBLE(ITE)),ITE,DEF,y
      ENDIF
 989  FORMAT("DEX(NLMIN)",   F8.2   )
 988  FORMAT("        ",  2F8.2,"        ",5F8.2   )
 986  FORMAT("        ",  2F8.2,"        ",5F8.2   )
 987  FORMAT(I8,7F8.2   )
c 986  FORMAT("Percent ",6F8.2)
 985  FORMAT("         ",F8.2,I4,2F8.2  )
 984  FORMAT("       ",F17.2,"   ",    F8.2  )

      END

      SUBROUTINE LL21_(DX,NX,XNORM)
      IMPLICIT DOUBLE PRECISION (A,C-H,O-U,W-Z),LOGICAL(B)
      PARAMETER (NNARR=299,NNLEV=9)
      DIMENSION VWORK(1),KWORK(1)
      DIMENSION DX(*)
```

```
      COMMON          NWORK,IWORK,IWMAX,L(NNARR),DWORK(1)
      EQUIVALENCE (DWORK(1),VWORK(1),KWORK(1))
      COMMON /ERRCTL/ IER,ICHECK
      SAVE /ERRCTL/
      include "jcommons.f"
C
c      IF (ICHECK.EQ.999) CALL OTRC('LL21  ','01/02/89')
C
      XNORM=0
      do i=1,NX
         XNORM=XNORM+DX(i)**2 * DWORK(L(KMASS(1))+i-1)
      enddo
c ----
c      CALL LSP1(DX,DX,NX,XNORM)
c ----
      XNORM=SQRT(XNORM)
      END




      SUBROUTINE YEX3(DX,DB,DD,DR,NEQ,DPREC)
c      SUBROUTINE YEX3(DX,DB,DD,DR,NEQ)

C
      IMPLICIT DOUBLE PRECISION (A,C-H,O-U,W-Z),LOGICAL(B)
      PARAMETER (NNARR=299,NNLEV=9)
      CHARACTER SUB*6,FMT*15,CPARAM*120
C
      DIMENSION VWORK(1),KWORK(1)
      DIMENSION DX(*),DB(*),DD(*),DR(*)
      COMMON /ERRCTL/ IER,ICHECK
      COMMON /CHAR/   SUB,FMT(3),CPARAM
      COMMON /MGPAR/  ILEV,NLEV,NLMIN,NLMAX,
     *                ICYCLE,KPRSM(NNLEV),KPOSM(NNLEV)
      COMMON /MGPAR0/ DOMPOS(NNLEV),DOMPRS(NNLEV),OMEX,EPSEX,
     *                KIPOSM(NNLEV),KIPRSM(NNLEV),IEX,NITEX,IELE
      COMMON /OUTPUT/ M,MT,MKEYB,MTERM,MERR,MPROT,MSYS,MTRC,IRECL8
      COMMON /XYPAR/  DXYPAR(NNARR),KXYPAR(NNARR)
      COMMON          NWORK,IWORK,IWMAX,L(NNARR),DWORK(1)
      EQUIVALENCE (DWORK(1),VWORK(1),KWORK(1))
      SAVE /ERRCTL/,/CHAR/,/MGPAR/,/MGPAR0/,/OUTPUT/,/XYPAR/
C
      EXTERNAL DPREC

      SUB='YEX3'
C
      LLA  =KXYPAR(100+4*ILEV-3)
      LDIA =KXYPAR(100+4*ILEV-2)
      LDIAS=KXYPAR(100+4*ILEV-1)
      NDIA =KXYPAR(100+4*ILEV)

      M0=M
      M=0
C
      IF (IEX.EQ.1) THEN
C
```

```
        IREQ=4*NEQ
        IF (OMEX.LT.0D0) IREQ=3*NEQ
        IREQ=MAX0(IREQ,4)
        CALL ZNEW(IREQ,1,LWORK,'WORKCG')
        IF (IER.NE.0) GOTO 99999
        L1=L(LWORK)
        L2=L1+NEQ
        L3=L2+NEQ
        L4=L3+NEQ
        IF (OMEX.LT.0D0) L4=L1
C
         ITE=0  ! statt IE317
         ITE=2

        CALL IE313h(DWORK(LLA),KWORK(LDIA),KWORK(LDIAS),(NDIA),
     *             DX,DB,NEQ,NITEX,ITE,EPSEX,OMEX,
     *             DWORK(L1),DWORK(L2),DWORK(L3),DWORK(L4),DPREC)
         IF (IER.NE.0) GOTO 99999
         WRITE(*,*)' IE313 iterations=',ITE
C
        IER1=IER
        CALL ZDISP(0,LWORK,'WORKCG')
        IER=IER1
C=======================================================
       ELSE IF (IEX.EQ.0) THEN ! EXACT GAUSS LU
        CALL SLVEX(DX,DB,NEQ)
C
       ENDIF
C
       END
C **********************************************
       SUBROUTINE SLVEX(DX,DB,NEQ)
       IMPLICIT DOUBLE PRECISION (A,C-H,O-U,W-Z),LOGICAL(B)
       PARAMETER (NNARR=299,NNLEV=9)
       CHARACTER SUB*6,FMT*15,CPARAM*120
        DOUBLE PRECISION timmat
       INTEGER LMATEX,LPIVOT
C
       DIMENSION VWORK(1),KWORK(1)
       DIMENSION DX(*),DB(*)
       COMMON /ERRCTL/ IER,ICHECK
       COMMON /CHAR/   SUB,FMT(3),CPARAM
       COMMON /MGPAR/  ILEV,NLEV,NLMIN,NLMAX,
      *                ICYCLE,KPRSM(NNLEV),KPOSM(NNLEV)
       COMMON /MGPAR0/ DOMPOS(NNLEV),DOMPRS(NNLEV),OMEX,EPSEX,
      *                KIPOSM(NNLEV),KIPRSM(NNLEV),IEX,NITEX,IELE
       COMMON /OUTPUT/ M,MT,MKEYB,MTERM,MERR,MPROT,MSYS,MTRC,IRECL8
       COMMON /XYPAR/  DXYPAR(NNARR),KXYPAR(NNARR)
       include "jcommons.f"
       COMMON          NWORK,IWORK,IWMAX,L(NNARR),DWORK(1)
       EQUIVALENCE (DWORK(1),VWORK(1),KWORK(1))
       SAVE /ERRCTL/,/CHAR/,/MGPAR/,/MGPAR0/,/OUTPUT/,/XYPAR/,/John/
Cc *****************INIT.MATRIX for LAPACK: XM013*******************
       III=100+(NLMIN-1)*4
       LLA  =(KXYPAR(III+1))
       LDIA =(KXYPAR(III+2))
       LDIAS=(KXYPAR(III+3))
```

```
      NDIA =KXYPAR(III+4)
      NSUB=ABS(KWORK(LDIA+1))
      NSUPER=NSUB
      LDA=2*NSUB+NSUPER+1 ! =3*(2**NLMIN+2)+1
Cc ************************************************************
      if (INIMAT.ne.1) THEN
         CALL ZTIME(T1)
      CALL ZNEW(LDA*NEQ,1,LMATEX,'DMAT')
      CALL ZNEW(NEQ,3,IPIVOT,'IPIVOT')
C ====LU=============================================
      CALL ZTIME(w1)
      CALL TRNSGB(NEQ,DWORK(LLA),KWORK(LDIA),LDIAS,NDIA,
     +  DWORK(L(LMATEX)),LDA,NSUB,NSUPER)
      INFO=0
      CALL DGBTRF(NEQ,NEQ,NSUB,NSUPER,DWORK(L(LMATEX)),LDA,
     +  KWORK(L(IPIVOT)),INFO)
      CALL ZTIME(w2)
      write(*,*)'Time for LU factorisation:',w2-w1
      INIMAT=1 ! 1st time only
         CALL ZTIME(T2)
         timmat=T2-T1
c      write(*,*)'Time fur Matrix aufbau+LU',T2-T1
      ENDIF ! INIMAT
C ************************************************************
 10   NRHS=1
      LDB=NEQ
      CALL DCOPY(NEQ,DB,1,DX,1)
C ====SOLVE==========================================================
      CALL ZTIME(w1)
      CALL DGBTRS('N',NEQ,NSUB,NSUPER,NRHS,DWORK(L(LMATEX)),LDA,
     +           KWORK(L(IPIVOT)),DX,LDB,INFO)
      CALL ZTIME(w2)
C =====================================================================
      END
C ---------------------------------------------------------------
      include 'transgb.f'




      SUBROUTINE YMAX13(DX,DAX,NEQ,A1,A2,IRES)
      IMPLICIT DOUBLE PRECISION (A,C-H,O-U,W-Z),LOGICAL(B)
      CHARACTER SUB*6,FMT*15,CPARAM*120
C
      PARAMETER (NNARR=299,NNLEV=9)
      DIMENSION VWORK(1),KWORK(1)
      DIMENSION DX(*),DAX(*)
      COMMON /ERRCTL/ IER,ICHECK
      COMMON /CHAR/   SUB,FMT(3),CPARAM
      COMMON /MGPAR/  ILEV,NLEV,NLMIN,NLMAX,
     *                ICYCLE,KPRSM(NNLEV),KPOSM(NNLEV)
      COMMON /XYPAR/  DXYPAR(NNARR),KXYPAR(NNARR)
      COMMON          NWORK,IWORK,IWMAX,L(NNARR),DWORK(1)
      include "jcommons.f"
      EQUIVALENCE (DWORK(1),VWORK(1),KWORK(1))
      SAVE /ERRCTL/,/CHAR/,/MGPAR/,/XYPAR/
      EXTERNAL band16,band8,band4,band2,band1
```

```
C
      SUB='YMAX13'
c     IF (ICHECK.GE.998) CALL OTRC('YMAX17','08/25/90')
      IER=0
C
C
      LLA  =KXYPAR(100+4*ILEV-3)
      LDIA =KXYPAR(100+4*ILEV-2)
      LDIAS=KXYPAR(100+4*ILEV-1)
      NDIA =KXYPAR(100+4*ILEV)
C
C      DAX := A1*DA*DX + A2*DAX
C
      IF(IMV.eq.0) THEN
      CALL LAX13(DWORK(LLA),KWORK(LDIA),KWORK(LDIAS),
     *     (NDIA),NEQ,DX,DAX,A1,A2)
C
      ELSE IF(IMV.eq.1) THEN
      CALL dgblmv(NEQ,DWORK(LLA),KWORK(LDIA),KWORK(LDIAS),NDIA,
     *       A1 ,DX, 1  , A2 ,DAX, 1  ,DAX)
C
      ELSE IF(IMV.eq.2) THEN
           incx=1
      CALL dgbmv2(NEQ,DWORK(LLA),KWORK(LDIA),KWORK(LDIAS),NDIA,
     *       A1 ,DX,incx , A2 ,DAX,incx,DAX)
C
      ELSE IF(IMV.eq.3) THEN
         IF(IBAND.eq.16)THEN
      CALL dgbmv3(NEQ,DWORK(LLA),KWORK(LDIA),KWORK(LDIAS),NDIA,
     *       A1 ,DX, 1   ,A2 ,DAX, 1  ,band16)
         ELSE IF(IBAND.eq.8)THEN
      CALL dgbmv3(NEQ,DWORK(LLA),KWORK(LDIA),KWORK(LDIAS),NDIA,
     *       A1 ,DX, 1   ,A2 ,DAX, 1  ,band8)
         ELSE IF(IBAND.eq.4)THEN
      CALL dgbmv3(NEQ,DWORK(LLA),KWORK(LDIA),KWORK(LDIAS),NDIA,
     *       A1 ,DX, 1   ,A2 ,DAX, 1  ,band4)
         ELSE IF(IBAND.eq.2)THEN
      CALL dgbmv3(NEQ,DWORK(LLA),KWORK(LDIA),KWORK(LDIAS),NDIA,
     *       A1 ,DX, 1   ,A2 ,DAX, 1  ,band2)
         ELSE IF(IBAND.eq.1)THEN
      CALL dgbmv3(NEQ,DWORK(LLA),KWORK(LDIA),KWORK(LDIAS),NDIA,
     *       A1 ,DX, 1   ,A2 ,DAX, 1  ,band1)
         ELSE
            write(*,*) "IBAND is of 1,2,4,8,16"
            stop
         ENDIF
      ENDIF
C
      END




C     smoothing
      SUBROUTINE ysm3(DX,DB,DD,DR,NEQ,NIT,IRES,OMEGA,MV,PREC)
```

```
      IMPLICIT DOUBLE PRECISION (A,C-H,O-U,W-Z),LOGICAL(B)
      PARAMETER (NNARR=299,NNAB=21,NNLEV=9,NNWORK=7000000)
      DIMENSION DX(*),DB(*),DD(*),DR(*)
      DIMENSION VWORK(1),KWORK(1)
      double precision OMEGA
      COMMON /ERRCTL/ IER,ICHECK
      COMMON /MGPAR/  ILEV,NLEV,NLMIN,NLMAX,
     *                ICYCLE,KPRSM(NNLEV),KPOSM(NNLEV)
      COMMON /MGPAR0/ DOMPOS(NNLEV),DOMPRS(NNLEV),OMEX,EPSEX,
     *                KIPOSM(NNLEV),KIPRSM(NNLEV),IEX,NITEX,IELE
      include "jcommons2.f" ! BCMG - boundary,matrizen
      include "jcommons.f"
      COMMON          NWORK,IWORK,IWMAX,L(NNARR),DWORK(1)
      EQUIVALENCE (DWORK(1),VWORK(1),KWORK(1))
      SAVE /ERRCTL/,/BCMG/
      EXTERNAL MV,PREC
C
      DO  ITE=1,NIT
C        Defect
         IF(IRES.eq.0)THEN
            CALL LCP1(DB,DR,NEQ) ! DR=DB
            CALL MV(NEQ,-1D0,DX,1,1D0,DR,1,DR)! Defect
         ENDIF
         IRES=0
C        C^-1*Defect
         CALL PREC(DX,DB,DD,DR,NEQ,ITE,OMEGA)
C        dx=dx+omega*dr
         CALL LLC1(DR,DX,NEQ,OMEGA,1D0)! dx=dx+omega*dr
      ENDDO
C
      END




C ************************************************************
      SUBROUTINE PROLONG(x,y)
C     Gegeben: vektor x an Level ILEV
C     Ausgabe: vector y an Level ILEV+1

      PARAMETER (NNVE=4,NNARR=299)
      DIMENSION VWORK(1),KWORK(1)
      COMMON          NWORK,IWORK,IWMAX,L(NNARR),DWORK(1)
      COMMON /TRIAA/  LCORVG,LCORMG,LVERT,LMID,LADJ,LVEL,LMEL,LNPR,LMM,
     *                LVBD,LEBD,LBCT,LVBDP,LMBDP
      EQUIVALENCE (DWORK(1),VWORK(1),KWORK(1))

      INTEGER i,k,o,q,r,s,level,m
      DOUBLE PRECISION x(*),y(*),Q2,Q4
      PARAMETER (Q2=0.5D0, Q4=0.25D0)
      PARAMETER (NNLEV=9)
      COMMON /MGPAR/  ILEV,NLEV,NLMIN,NLMAX,
     *                ICYCLE,KPRSM(NNLEV),KPOSM(NNLEV)
      include "jcommons.f"

      level=ILEV-1
      k=2**(level+1)            ! #nodes on row -1 ! ILEV=2 k=8
```

```
      m=(k+1)                   !                 !         m=9
      o=m*2-1                   ! step            !         o=17
      q=2**level+1              ! #nodes on row   !         q=5
      r=q-1                     !                 !         r=4
      s=o+1                     ! 2* #nodes on row!         s=18
      CALL DSCAL(m**2,0.0D0,y,1)
C     first: old points to new points
      DO i=1,q
          CALL DCOPY(q,x(i+r*(i-1)),1,y(i+o*(i-1)),2)
      ENDDO
C     New points in horzontal direction
      DO j=1,q-1
         p=j*2
          CALL DAXPY(q,Q2,x(j-0),q,y(p),s)
          CALL DAXPY(q,Q2,x(j+1),q,y(p),s)
C     new points in vertical direction
         p=m+1+(j-1)*s
         CALL DAXPY(q,Q2,y(p-m),2,y(p),2)
         CALL DAXPY(q,Q2,y(p+m),2,y(p),2)
      ENDDO
C     new 'mid points'
      DO i=1,r
         j=k+3+(i-1)*s
         CALL DAXPY(r,Q4,y(j-k-2),2,y(j),2)
         CALL DAXPY(r,Q4,y(j-k  ),2,y(j),2)
         CALL DAXPY(r,Q4,y(j+k  ),2,y(j),2)
         CALL DAXPY(r,Q4,y(j+k+2),2,y(j),2)
      ENDDO
 42   END
C *************************************************************
      SUBROUTINE RESTRICT(x,y)
C     GEGEBEN: vektor x an level
C     AUSGABE: vektor y an level-1

      INTEGER i,k,m,level,o,q,r,s
      DOUBLE PRECISION x(*),y(*),Q2,Q4
      PARAMETER (NNLEV=9)
      PARAMETER (Q2=0.5D0, Q4=0.25D0)
      COMMON /MGTRD/  KNEL(NNLEV),KNVT(NNLEV),KNMT(NNLEV),
     *                KNVEL(NNLEV),KNVBD(NNLEV)
      COMMON /MGPAR/  ILEV,NLEV,NLMIN,NLMAX,
     *                ICYCLE,KPRSM(NNLEV),KPOSM(NNLEV)
      level=ILEV+2
      k=2**(level-1)            ! #nodes on row -1
      m=(k+1)                   ! rows
      o=m*2-1                   ! step
      q=2**(level-1)+1          ! #nodes on row
      r=q-1                     !
      s=int(m/2)+1

      DO i=1,s
         CALL DCOPY(s,x(1+2*m*(i-1)),2,y(1+s*(i-1)),1)
      ENDDO
      DO i=1,s-1
C         .5 * side-neighbors
       CALL DAXPY(s,Q2, x(1+2*m*(i-1)+m) ,2, y(1+s*(i-1)) ,1 )
       CALL DAXPY(s,Q2, x(1+2*m*(i-1)+m) ,2, y(1+s*(i  )) ,1 )
```

```
       CALL DAXPY(s,Q2, x(2+2*(i-1)) ,2*m,y(    (i  )) ,s)
       CALL DAXPY(s,Q2, x(2+2*(i-1)) ,2*m,y(    (i+1)) ,s)
C         .25* diag-neighbors
       CALL DAXPY(s-1,Q4,x(2+2*m*(i-1)+m),2, y(1+s*(i-1)) ,1 )
       CALL DAXPY(s-1,Q4,x(2+2*m*(i-1)+m),2, y(2+s*(i-1)) ,1 )
       CALL DAXPY(s-1,Q4,x(2+2*m*(i-1)+m),2, y(1+s*(i-0)) ,1 )
       CALL DAXPY(s-1,Q4,x(2+2*m*(i-1)+m),2, y(2+s*(i-0)) ,1 )
      ENDDO
C
      END




      SUBROUTINE YMBC013(DX,NEQ)
      IMPLICIT DOUBLE PRECISION (A,C-H,O-U,W-Z),LOGICAL(B)
      CHARACTER SUB*6,FMT*15,CPARAM*120
C
      PARAMETER (NNARR=299,NNLEV=9)
      DIMENSION VWORK(1),KWORK(1)
      DIMENSION DX(*)
      COMMON /ERRCTL/ IER,ICHECK
      COMMON /CHAR/   SUB,FMT(3),CPARAM
      COMMON /TRIAD/  NEL,NVT,NMT,NVE,NVEL,NBCT,NVBD
      COMMON /MGTRA/  KLCVG(NNLEV),KLCMG(NNLEV),KLVERT(NNLEV),
     *                KLMID(NNLEV),KLADJ(NNLEV),KLVEL(NNLEV),
     *                KLMEL(NNLEV),KLNPR(NNLEV),KLMM(NNLEV),
     *                KLVBD(NNLEV),KLEBD(NNLEV),KLBCT(NNLEV),
     *                KLVBDP(NNLEV),KLMBDP(NNLEV)
      COMMON /MGPAR/  ILEV,NLEV,NLMIN,NLMAX,
     *                ICYCLE,KPRSM(NNLEV),KPOSM(NNLEV)
      include "jcommons2.f" ! BCMG - boundary,matrizen
      COMMON          NWORK,IWORK,IWMAX,L(NNARR),DWORK(1)
      EQUIVALENCE (DWORK(1),VWORK(1),KWORK(1))
      SAVE /ERRCTL/,/CHAR/,/TRIAD/,/MGPAR/,/MGTRA/
C
      IER=0
C     set boundary vals to 0 if in bndry array
      LBCMG=L(KLBCMG(ILEV))
      NBCMG=KNBCMG(ILEV)
      DO IBCMG=1,NBCMG
         if( KWORK(LBCMG+IBCMG-1) .ne. 0)  DX(KWORK(LBCMG+IBCMG-1))=0D0
      ENDDO
C
      END




      subroutine dgbmv3(n,d,index,offs,lindex,
     *                  alpha,x,incx,beta,y,incy,band)
      implicit none
c
c
      double precision x(*),y(*),d(*),alpha,beta
      integer incx,incy,n,index(*),lindex
      integer bindex,si,di,offset,len,offs(*)
```

```
      external dscal,band
c
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c

      PRINT *,"INCX != 1 yet not implemented"
      return
c
c
20    offset=1

      if (beta.eq.0.0D0) then
         call dcopy(n,0.0D0,0,y,incy)
      else
         if ((beta/alpha).ne.1.0) then
           call dscal(n,beta/alpha,y,incy)
         endif
      endif

      do 1100,bindex=1,lindex

      len=n-abs(index(bindex))

      if (index(bindex).gt.0) THEN
          si=index(bindex)+1
          di=1
      else
          di=-(index(bindex)-1)
          si=1
      end if

      call band(n,len,y(di),x(si),d(offset))

  offset=offset+len

1100  continue

      if (alpha.ne.1.0D0) then
         call dscal(n,alpha,y,incy)
  endif

  end

C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  subroutine band16(n,ulen,y,x,d)
  implicit none

  integer len,ulen,n,mp1,m,i,offset
  double precision y(*),x(*),d(*)

  offset=1
  len=ulen
```

```
      m = mod(len,16)
      if( m .eq. 0 ) go to 1040
      do 1030 i = 1,m
        y(i) = y(i) + d(offset)*x(i)
        offset=offset+1
 1030 continue
      len=len-m
      if( n .lt. 16 ) go to 1060
 1040 mp1 = m + 1
      do 1050 i = mp1,len,16
        y(i) = y(i) + d(offset)*x(i)
        y(i+1) = y(i+1) + d(offset+1)*x(i+1)
        y(i+2) = y(i+2) + d(offset+2)*x(i+2)
        y(i+3) = y(i+3) + d(offset+3)*x(i+3)
        y(i+4) = y(i+4) + d(offset+4)*x(i+4)
        y(i+5) = y(i+5) + d(offset+5)*x(i+5)
        y(i+6) = y(i+6) + d(offset+6)*x(i+6)
        y(i+7) = y(i+7) + d(offset+7)*x(i+7)
        y(i+8) = y(i+8) + d(offset+8)*x(i+8)
        y(i+9) = y(i+9) + d(offset+9)*x(i+9)
        y(i+10) = y(i+10) + d(offset+10)*x(i+10)
        y(i+11) = y(i+11) + d(offset+11)*x(i+11)
        y(i+12) = y(i+12) + d(offset+12)*x(i+12)
        y(i+13) = y(i+13) + d(offset+13)*x(i+13)
        y(i+14) = y(i+14) + d(offset+14)*x(i+14)
        y(i+15) = y(i+15) + d(offset+15)*x(i+15)

      offset=offset+16
 1050 continue
 1060 continue

      end
```

```
      SUBROUTINE refine(NFINE,DCORVG,KVERT,PARX,PARY)
      INTEGER NFINE,i,j,k,ll
      DOUBLE PRECISION x(4),y(4),ps1,ps2,ps3,qs1,qs2,qs3
      DOUBLE PRECISION xleft,xright,ydown,yup
      IMPLICIT DOUBLE PRECISION (A,C-H,O-U,W-Z),LOGICAL(B)

      DOUBLE PRECISION dlen,dpoint(2),dfine(3),dmin(2)
      integer idim,nlen,lline,lline2

      CHARACTER SUB*6,FMT*15,CPARAM*120
      PARAMETER (NNARR=299,NNVE=4)
      DIMENSION DCORVG(2,*)
      DIMENSION KVERT(NNVE,*)
      COMMON /OUTPUT/ M,MT,MKEYB,MTERM,MERR,MPROT,MSYS,MTRC,IRECL8
      COMMON /ERRCTL/ IER,ICHECK
      COMMON          NWORK,IWORK,IWMAX,L(NNARR),DWORK(1)
      COMMON /CHAR/   SUB,FMT(3),CPARAM
      COMMON /TRIAD/  NEL,NVT,NMT,NVE,NVEL,NBCT,NVBD
      include "jcommons.f"
```

```
      EXTERNAL PARX,PARY
      SAVE /OUTPUT/,/ERRCTL/,/CHAR/,/TRIAD/
C -----------------------------------------------------------------------
      ll=2**NFINE
      NEL=ll**2
      NVT=(ll+1)**2
C     KVERT: row-wise numbering !
      j=1
      DO i=1,NEL
         KVERT(1,i)=j
         KVERT(2,i)=j+1
         KVERT(3,i)=j+1+ll+1
         KVERT(4,i)=j  +ll+1
         j=j+1
         if(mod(j,ll+1).eq.0) j=j+1
      ENDDO
C -----------------------------------------------------------------------
C     DCORVG: Given a grid with four points ...
      x(1)=PARX(0D0,1)
      y(1)=PARY(0D0,1)
      x(2)=PARX(1D0,1)
      y(2)=PARY(1D0,1)
      x(3)=PARX(3D0,1)
      y(3)=PARY(3D0,1)
      x(4)=PARX(2D0,1)
      y(4)=PARY(2D0,1)
C -----------------------------------------------------------------------
      dmin (1)=0
      dmin (2)=0
      idim=1
      nlen=2**NFINE
C -------------------------------------------------------------X------
      dpoint(1)=0.D0
      dpoint(2)=1.D0
      dlen=dpoint(2)-dpoint(1)
      dfine(1)=ps1
      dfine(2)=ps2
      dfine(3)=ps3
      if(nlen.gt.2)then         ! use grid function
         CALL ZNEW(2*(ll+1),1,lline2,"lline2")

         if(Idir2.eq.1)THEN
            call sgrid_refine(DWORK(L(lline2)),
     *           dpoint,dlen,nlen,dfine,dmin,idim)
         else
            call susy2(DWORK(L(lline2)),
     *           dpoint,dlen,nlen,dfine,dmin,idim)
         ENDIF
      endif
C -------------------------------------------------------------Y------
      dpoint(1)=0.D0
      dpoint(2)=1.D0
      dfine(1)=qs1
      dfine(2)=qs2
      dfine(3)=qs3
      if(nlen.gt.2)then         ! use grid function
         CALL ZNEW(2*(ll+1),1,lline,"lline")
```

```
         if(1.eq.1)THEN
            call sgrid_refine(DWORK(L(lline)),
     *           dpoint,dlen,nlen,dfine,dmin,idim)
         else
            call susy2(DWORK(L(lline)),
     *           dpoint,dlen,nlen,dfine,dmin,idim)
         ENDIF
       endif
C ---------------------------------------------------------------------
c     34
c     12
C ---------------------------------------------------------------------
      DO i=1,ll+1
      DO j=1,ll+1
         k=i+(j-1)*(ll+1)
         xleft=x(1)+DWORK(L(lline)+(j-1)*2)*(x(3)-x(1))
         xright=x(2)+DWORK(L(lline)+(j-1)*2)*(x(4)-x(2))
         ydown=y(1)+DWORK(L(lline2)+(i-1)*2)*(y(2)-y(1))
         yup=y(3)+DWORK(L(lline2)+(i-1)*2)*(y(4)-y(3))
c         write(*,*) xleft,xright,ydown,yup
         DCORVG(1,k)=xleft+(xright-xleft) *DWORK(L(lline2)+(i-1)*2)
         DCORVG(2,k)=ydown+(yup-ydown)    *DWORK(L(lline )+(j-1)*2)
      ENDDO
      ENDDO
C ---------------------------------------------------------------------
      if(NFINE.eq.2)then
      DO i=1,ll+1
      DO j=1,ll+1
         k=i+(j-1)*(ll+1)
c         write(*,*) i,j,k,DCORVG(1,k),DCORVG(2,k)
      ENDDO
      ENDDO
      endif

 2    END




C *****************************************************
C     prepare tridiagonal preconditioner
      SUBROUTINE INITRI(KLA,KLDIAA,KLDIAS,KNDIAA,KNEQ)
c      IMPLICIT NONE
       IMPLICIT DOUBLE PRECISION(A,C-H,O-U,W-Z),LOGICAL(B)
      PARAMETER (NNARR=299,NNAB=21,NNLEV=9,NNWORK=7000000)
      PARAMETER (NBLOCA=1,NBLOCF=1)
      DIMENSION KLA(NBLOCA,NNLEV)
      DIMENSION LA(NBLOCA)
      DIMENSION VWORK(1),KWORK(1)
      DIMENSION KNEQ(NNLEV)
      DIMENSION KLDIAA(NNLEV),KLDIAS(NNLEV),KNDIAA(NNLEV)
      COMMON          NWORK,IWORK,IWMAX,L(NNARR),DWORK(NNWORK)
      COMMON /ERRCTL/ IER,ICHECK
      COMMON /MGPAR/  ILEV,NLEV,NLMIN,NLMAX,
     *                ICYCLE,KPRSM(NNLEV),KPOSM(NNLEV)
```

```
      include "jcommons.f"
      include "jcommons2.f" ! BCMG - boundary,matrizen
      SAVE /BCMG/
      EQUIVALENCE (DWORK(1),VWORK(1),KWORK(1))

c      write(*,*)'INITRI ilev=:',ilev
      ILEV2=ILEV
c      DO ILEV2=NLMIN+1,NLMAX
         NEQ=KNEQ(ILEV2)
        LA(1)=KLA(1,ILEV2)
        I=(NEQ-1)+NEQ+(NEQ-1)+(NEQ-2)
        call znew( I ,1,LPREC,'LPREC') ! MEM for Tri-matrix
        call znew( NEQ,3,LPIV,'LPIV')  !     and pivot
        LDIA=KLDIAA( ILEV2)
        LDIAS=KLDIAS(ILEV2)
        NDIA=KNDIAA( ILEV2)
        CALL DCOPY(NEQ,DWORK(L(LA(1))),1,DWORK(L(LPREC)),1) !diag0
        DO I=1,NDIA
           if(KWORK(L(LDIA)+i-1).eq.-1) THEN
             CALL DCOPY(NEQ-1,DWORK(L(LA(1))+KWORK(L(LDIAS)+i-1)-1),1,
     *                       DWORK(L(LPREC)+NEQ),1)
            else if(KWORK(L(LDIA)+i-1).eq.1) THEN
             CALL DCOPY(NEQ-1,DWORK(L(LA(1))+KWORK(L(LDIAS)+i-1)-1),1,
     *                       DWORK(L(LPREC)+NEQ*2-1),1)
           endif
        ENDDO
         LDIAG=L(LPREC)
         LOWER=L(LPREC)
         LUP1=L(LPREC)
         LUP2=L(LPREC)
         IPIVOT=L(LPIV)
         INFO=0
C--------------------------------
 2        if (IBLOCK.ne.1)THEN
          CALL DGTTRF(NEQ,DWORK(LOWER+NEQ),DWORK(LDIAG),
     *     DWORK(LUP1+NEQ*2-1),DWORK(LUP2+NEQ*3-2),KWORK(IPIVOT),INFO)
          IF(INFO.ne.0) THEN
             write(*,*)"TRIDAIG ERROR in ttest.f, INFO=",INFO
             stop
          ENDIF
         else
C ************************
         Dbllen=SQRT(DBLE(NEQ))    ! length of a line/block
         Ibllen=INT(Dbllen)
         DO I=1,Ibllen ! solve block-wise  TO DO !!!!!!!!!!!!!!!
           Ioffs=(I-1)*Ibllen
           CALL DGTTRF(Ibllen,DWORK(LOWER+NEQ+Ioffs),DWORK(LDIAG+Ioffs),
     *     DWORK(LUP1+NEQ*2-1+Ioffs),DWORK(LUP2+NEQ*3-2+Ioffs),
     *     KWORK(IPIVOT+Ioffs),INFO)
          ENDDO
C ************************
 3     endif
        KLPREC(ILEV2)=LPREC
        KLPIV(ILEV2)=LPIV
c      ENDDO
      END
C ***************************************************************
```

```
 SUBROUTINE INITRI2(KLA,KLDIAA,KLDIAS,KNDIAA,KNEQ)
 IMPLICIT DOUBLE PRECISION(A,C-H,O-U,W-Z),LOGICAL(B)
 PARAMETER (NNARR=299,NNAB=21,NNLEV=9,NNWORK=7000000)
 PARAMETER (NBLOCA=1,NBLOCF=1)
 DIMENSION KLA(NBLOCA,NNLEV)
 DIMENSION LA(NBLOCA)
 DIMENSION VWORK(1),KWORK(1)
 DIMENSION KNEQ(NNLEV)
 DIMENSION KLDIAA(NNLEV),KLDIAS(NNLEV),KNDIAA(NNLEV)
 INTEGER offs1,offs2,LLEN,ibllen
 integer offdm,offdp
 double precision dbllen
 COMMON          NWORK,IWORK,IWMAX,L(NNARR),DWORK(NNWORK)
 COMMON /ERRCTL/ IER,ICHECK
 COMMON /MGPAR/  ILEV,NLEV,NLMIN,NLMAX,
*                ICYCLE,KPRSM(NNLEV),KPOSM(NNLEV)
 include "jcommons.f" ! BCMG - boundary,matrizen
 include "jcommons2.f" ! BCMG - boundary,matrizen
 SAVE /BCMG/
 EQUIVALENCE (DWORK(1),VWORK(1),KWORK(1))

    ILEV2=ILEV
    NEQ=KNEQ(ILEV2)
    LLEN=INT(SQRT(DBLE(NEQ)))
    LA(1)=KLA(1,ILEV2)
    I=(NEQ-1)+NEQ+(NEQ-1)+(NEQ-2)
    call znew(I  ,1,LPREC,'LPREC') ! MEM for Tri-matrix
    call znew(NEQ,3,LPIV,'LPIV')  !     and pivot
    LDIA=KLDIAA( ILEV2)
    LDIAS=KLDIAS(ILEV2)
    NDIA=KNDIAA( ILEV2)

    if(NDIA.eq.9)then
       offdm=2
       offdp=7
    elseif(NDIA.eq.5)then
       offdm=1
       offdp=4
    else
       write(*,*)'TRI2 is for NDIA in (5,9)'
       stop
    endif

    DO I=1,NEQ
       J=ITINDX(I,ILEV2)
       IF(I.LT.J) THEN
          DWORK(L(LPREC)+I-1)=DWORK(L(LA(1))+J-1)!diag0
          DWORK(L(LPREC)+J-1)=DWORK(L(LA(1))+I-1)!diag0
       ELSE IF(j.EQ.I)THEN
          DWORK(L(LPREC)+I-1)=DWORK(L(LA(1))+I-1)!diag0
       ENDIF
    ENDDO  ! I
    DO I=1,NEQ-LLEN ! length of subdiag
          offs2=I-1
          offs1=ITINDX(I,ILEV2)
          DWORK(L(LPREC)+NEQ+offs1-1  )=
*             DWORK(L(LA(1))+KWORK(L(LDIAS)+offdm)-1+offs2)!diag-1
```

```
               DWORK(L(LPREC)+NEQ*2+offs1-2)=
     *             DWORK(L(LA(1))+KWORK(L(LDIAS)+offdp)-1+offs2)!diag1
          ENDDO
          LDIAG=L(LPREC)
          LLOWER=L(LPREC)
          LUP1=L(LPREC)
          LUP2=L(LPREC)
          IPIVOT=L(LPIV)
          INFO=0
C--------------------------------
C     Block solve technique
 2        if (IBLOCK.ne.1)THEN
             CALL DGTTRF(NEQ,DWORK(LLOWER+NEQ),DWORK(LDIAG),
     *            DWORK(LUP1+NEQ*2-1),DWORK(LUP2+NEQ*3-2),
     *            KWORK(IPIVOT),INFO)
             IF(INFO.ne.0) THEN
                write(*,*)"TRI2 ERROR, INFO=",INFO
                stop
             ENDIF
           else
             Dbllen=SQRT(DBLE(NEQ)) ! length of a line/block
             Ibllen=INT(Dbllen)
             DO I=1,Ibllen      ! solve block-wise  TO DO !!!!!!!!!!!!!!!
                Ioffs=(I-1)*Ibllen
                CALL DGTTRF(Ibllen,DWORK(LLOWER+NEQ+Ioffs),
     *               DWORK(LDIAG+Ioffs),DWORK(LUP1+NEQ*2-1+Ioffs),
     *               DWORK(LUP2+NEQ*3-2+Ioffs),
     *               KWORK(IPIVOT+Ioffs),INFO)
             ENDDO
           endif                ! blocking
C ***********************
 3       KLPREC(ILEV2+NNLEV)=LPREC
          KLPIV(ILEV2+NNLEV)=LPIV
C=====================================================
C      GSTRI2
C     For GSTri2, save L of permuted A (column-wise grid)
          IF(IPREC.eq.6 .or. IPREC.eq.7) THEN

            NEQ=KNEQ(ILEV2)
            LLEN=INT(SQRT(DBLE(NEQ)))
            LA(1)=KLA(1,ILEV2)
            I=(NEQ-Ibllen-1)+(NEQ-ibllen)+(NEQ-Ibllen-1)
            call znew(I  ,1,LPREC,'LPREC') ! MEM for Tri-matrix
            LDIA=KLDIAA( ILEV2)
            LDIAS=KLDIAS(ILEV2)
            NDIA=KNDIAA( ILEV2)
            Iup=0
            Idiag=0
            Ilow=0

            do I=1,NDIA
               j=KWORK(L(LDIA)+I-1)
               if(j.eq.-1)         Idiag=KWORK(L(LDIAS)+i-1)-1
               if(j.eq.(Ibllen-1))   Iup=KWORK(L(LDIAS)+i-1)
               if(j.eq.-(Ibllen+1)) Ilow=KWORK(L(LDIAS)+i-1)-1
            enddo
C
```

```
C ... L(i)=A(p(i))
C
            llprec=l(lprec)
            is1=NEQ-Ibllen
            is2=2*is1
            LLA=L(LA(1))
          do i=1,NEQ-IBLLEN-1

             DWORK(llprec+i-1+1)  =DWORK(LLA+Ilow+ITINDX(i,ILEV2)-1)
            DWORK(llprec+i-1+is1)=DWORK(LLA+idiag+ITINDX(i,ILEV2)-1)
            DWORK(llprec+i-1+is2)=DWORK(LLA+iup+ITINDX(i,ILEV2)-1)
          enddo
          KLPREC(ILEV2+NNLEV*2)=LPREC


       endif   !--------------GSTRI2-------------------------------
 60   END

C ***************************************************
C     ITINDX: return index of transposed grid for NODE
      INTEGER FUNCTION ITINDX(NODE,ILEV)
      INTEGER NODE,LLEN,i,j
      LLEN=(2**ILEV+1) ! number vertices on a row
      i=MOD(NODE-1,LLEN)+1
      j=(NODE-1)/LLEN+1
      ITINDX=(i-1)*LLEN+j
      return
      end
C ***************************************************




C ***************************************************
C     prepare tridiagonal preconditioner
      SUBROUTINE INIGSTRI(KLA,KLDIAA,KLDIAS,KNDIAA,KNEQ)
c      IMPLICIT NONE
       IMPLICIT DOUBLE PRECISION(A,C-H,O-U,W-Z),LOGICAL(B)
      PARAMETER (NNARR=299,NNAB=21,NNLEV=9,NNWORK=7000000)
      PARAMETER (NBLOCA=1,NBLOCF=1)
      DIMENSION KLA(NBLOCA,NNLEV)
      DIMENSION LA(NBLOCA)
      DIMENSION VWORK(1),KWORK(1)
      DIMENSION KNEQ(NNLEV)
      DIMENSION KLDIAA(NNLEV),KLDIAS(NNLEV),KNDIAA(NNLEV)
      COMMON          NWORK,IWORK,IWMAX,L(NNARR),DWORK(NNWORK)
      COMMON /ERRCTL/ IER,ICHECK
      COMMON /MGPAR/  ILEV,NLEV,NLMIN,NLMAX,
     *                ICYCLE,KPRSM(NNLEV),KPOSM(NNLEV)
      include "jcommons.f"
      include "jcommons2.f" ! BCMG - boundary,matrizen
      SAVE /BCMG/
      EQUIVALENCE (DWORK(1),VWORK(1),KWORK(1))

         ILEV2=ILEV
          NEQ=KNEQ(ILEV2)
         LA(1)=KLA(1,ILEV2)
```

```
         I=(NEQ-1)+NEQ+(NEQ-1)+(NEQ-2)
         call znew( I ,1,LPREC,'LPREC') ! MEM for Tri-matrix
         call znew( NEQ,3,LPIV,'LPIV')  !     and pivot
         LDIA=KLDIAA( ILEV2)
         LDIAS=KLDIAS(ILEV2)
         NDIA=KNDIAA( ILEV2)
         CALL DCOPY(NEQ,DWORK(L(LA(1))),1,DWORK(L(LPREC)),1) !diag0
         DO I=1,NDIA
            if(KWORK(L(LDIA)+i-1).eq.-1) THEN
              CALL DCOPY(NEQ,DWORK(L(LA(1))+KWORK(L(LDIAS)+i-1)-1),1,
     *                       DWORK(L(LPREC)+NEQ),1)
            endif
         ENDDO
          LDIAG=L(LPREC)
          LOWER=L(LPREC)
          LUP1=L(LPREC)
          LUP2=L(LPREC)
          IPIVOT=L(LPIV)
          INFO=0
 2        if (IBLOCK.ne.1)THEN
C ************************
          CALL DGTTRF(NEQ,DWORK(LOWER+NEQ),DWORK(LDIAG),
     *     DWORK(LUP1+NEQ*2-1),DWORK(LUP2+NEQ*3-2),KWORK(IPIVOT),INFO)
          IF(INFO.ne.0) THEN
             write(*,*)"TRIDAIG ERROR in ttest.f, INFO=",INFO
             stop
          ENDIF
         else
C ************************
         Dbllen=SQRT(DBLE(NEQ))    ! length of a line/block
         Ibllen=INT(Dbllen)
         DO I=1,Ibllen ! solve block-wise  TO DO !!!!!!!!!!!!!!!
           Ioffs=(I-1)*Ibllen
           CALL DGTTRF(Ibllen,DWORK(LOWER+NEQ+Ioffs),DWORK(LDIAG+Ioffs),
     *     DWORK(LUP1+NEQ*2-1+Ioffs),DWORK(LUP2+NEQ*3-2+Ioffs),
     *     KWORK(IPIVOT+Ioffs),INFO)
          ENDDO
C ************************
         endif
         KLPREC(ILEV2)=LPREC
         KLPIV(ILEV2)=LPIV
c      ENDDO
 24   format(5f5.2)
      END
C *****************************************************************
      SUBROUTINE INIGSTRI2(KLA,KLDIAA,KLDIAS,KNDIAA,KNEQ)
c     IMPLICIT NONE
      IMPLICIT DOUBLE PRECISION(A,C-H,O-U,W-Z),LOGICAL(B)
      PARAMETER (NNARR=299,NNAB=21,NNLEV=9,NNWORK=7000000)
      PARAMETER (NBLOCA=1,NBLOCF=1)
      DIMENSION KLA(NBLOCA,NNLEV)
      DIMENSION LA(NBLOCA)
      DIMENSION VWORK(1),KWORK(1)
      DIMENSION KNEQ(NNLEV)
      DIMENSION KLDIAA(NNLEV),KLDIAS(NNLEV),KNDIAA(NNLEV)
      INTEGER offs1,offs2,LLEN,ibllen
      double precision dbllen
```

```
      COMMON          NWORK,IWORK,IWMAX,L(NNARR),DWORK(NNWORK)
      COMMON /ERRCTL/ IER,ICHECK
      COMMON /MGPAR/  ILEV,NLEV,NLMIN,NLMAX,
     *                ICYCLE,KPRSM(NNLEV),KPOSM(NNLEV)
      include "jcommons.f" ! BCMG - boundary,matrizen
      include "jcommons2.f" ! BCMG - boundary,matrizen
      SAVE /BCMG/
      EQUIVALENCE (DWORK(1),VWORK(1),KWORK(1))


      ILEV2=ILEV
         NEQ=KNEQ(ILEV2)
         LLEN=INT(SQRT(DBLE(NEQ)))
         LA(1)=KLA(1,ILEV2)
         I=(NEQ-1)+NEQ+(NEQ-1)+(NEQ-2)
         call znew(I  ,1,LPREC,'LPREC') ! MEM for Tri-matrix
         call znew(NEQ,3,LPIV,'LPIV')  !     and pivot
         LDIA=KLDIAA( ILEV2)
         LDIAS=KLDIAS(ILEV2)
         NDIA=KNDIAA( ILEV2)
         IF(NDIA.ne.9) THEN
            write(*,*)"For Tri^2 NDIA should equal 9"
            write(*,*)"adi.f"
            stop !For Tri^2 NDIA should equal 9
         ENDIF
         DO I=1,NEQ
            J=ITINDX(I,ILEV2)
            IF(I.LT.J) THEN
               DWORK(L(LPREC)+I-1)=DWORK(L(LA(1))+J-1)!diag0
               DWORK(L(LPREC)+J-1)=DWORK(L(LA(1))+I-1)!diag0
            ELSE IF(j.EQ.I)THEN
               DWORK(L(LPREC)+I-1)=DWORK(L(LA(1))+I-1)!diag0
            ENDIF
         ENDDO  ! I
         DO I=1,NEQ-LLEN ! length of subdiag
               offs2=I-1
               offs1=ITINDX(I,ILEV2)
               DWORK(L(LPREC)+NEQ+offs1-1  )=
     *             DWORK(L(LA(1))+KWORK(L(LDIAS)+2)-1+offs2)!diag-1
         ENDDO
          LDIAG=L(LPREC)
          LLOWER=L(LPREC)
          LUP1=L(LPREC)
          LUP2=L(LPREC)
          IPIVOT=L(LPIV)
          INFO=0
C     Block solve technique
 2        if (IBLOCK.ne.1)THEN
          CALL DGTTRF(NEQ,DWORK(LLOWER+NEQ),DWORK(LDIAG),
     *     DWORK(LUP1+NEQ*2-1),DWORK(LUP2+NEQ*3-2),KWORK(IPIVOT),INFO)
          IF(INFO.ne.0) THEN
             write(*,*)"TRIDAIG ERROR in ttest.f, INFO=",INFO
             stop
          ENDIF
         else
         Dbllen=SQRT(DBLE(NEQ))    ! length of a line/block
         Ibllen=INT(Dbllen)
```

```
         DO I=1,Ibllen ! solve block-wise  TO DO !!!!!!!!!!!!!!!
            Ioffs=(I-1)*Ibllen
          CALL DGTTRF(Ibllen,DWORK(LLOWER+NEQ+Ioffs),
     *     DWORK(LDIAG+Ioffs),
     *     DWORK(LUP1+NEQ*2-1+Ioffs),DWORK(LUP2+NEQ*3-2+Ioffs),
     *     KWORK(IPIVOT+Ioffs),INFO)
          ENDDO ! ************************
         endif ! blocking
         KLPREC(ILEV2+NNLEV)=LPREC
         KLPIV( ILEV2+NNLEV)=LPIV
C     For GSTri2, save L of permuted A (column-wise grid)
         IF(IPREC.eq.6 .or. IPREC.eq.7 .or.
     *      IPREC.eq.12.or. IPREC.eq.13) THEN

           NEQ=KNEQ(ILEV2)
           LLEN=INT(SQRT(DBLE(NEQ)))
           LA(1)=KLA(1,ILEV2)
           I=(NEQ-Ibllen-1)+(NEQ-ibllen)+(NEQ-Ibllen-1)
           call znew(I  ,1,LPREC,'LPREC') ! MEM for Tri-matrix
           LDIA=KLDIAA( ILEV2)
           LDIAS=KLDIAS(ILEV2)
           NDIA=KNDIAA( ILEV2)
           Iup=0
           Idiag=0
           Ilow=0

           do I=1,NDIA
              j=KWORK(L(LDIA)+I-1)
              if(j.eq.-1)         Idiag=KWORK(L(LDIAS)+i-1)-1
              if(j.eq.(Ibllen-1))   Iup=KWORK(L(LDIAS)+i-1)
              if(j.eq.-(Ibllen+1)) Ilow=KWORK(L(LDIAS)+i-1)-1
           enddo
           if(Idiag.eq.0 .or. Iup.eq.0 .or. Ilow.eq.0)THEN
              write(*,*)'diags Iup,Idiag,Ilow not found',Iup,Idiag,Ilow
              stop
           endif
C
C ... L(i)=A(p(i))
C
          llprec=l(lprec)
          is1=NEQ-Ibllen
          is2=2*is1
          LLA=L(LA(1))
          do i=1,NEQ-IBLLEN-1
             DWORK(llprec+i-1)    =DWORK(LLA+Ilow+ITINDX(i,ILEV2)-1)
             DWORK(llprec+i-1+is1)=DWORK(LLA+idiag+ITINDX(i,ILEV2)-1)
             DWORK(llprec+i-1+is2)=DWORK(LLA+iup+ITINDX(i,ILEV2)-1)
          enddo

         KLPREC(ILEV2+NNLEV*2)=LPREC
         endif   !-----------------------------------------
 24   format(25F4.1)
 54   format(25I4)
      END
```