UNIVERSITÄT
BREMEN

Zentrum für Technomathematik

Fachbereich 3 – Mathematik und Informatik

Recursive mesh refinement in 3D

Gianfrancesco Martinico

Report 00-03

Berichte aus der Technomathematik

Report 00-03 **Februar 2000**

Recursive mesh refinement in 3D

Gianfrancesco Martinico

Abstract

In this paper a three-dimensional mesh generator is introduced. The mesh generator takes the works of Bänsch and Kossaczky [4, 8] as starting point. The developed method shows how to divide a regular domain into non degenerative tetrahedra, using a recursive approach. It is possible to invert the process and to restore previously refined tetrahedra. We will also show how the used data structure allows for the studying of irregular domains.

Keywords

Finite Element Method, domain discretization, adaptivity, conformity, mesh refinement , mesh coarsen.

1. Introduction

Many physical problems in a wide range of science are modeled using partial differential equations (PDEs). For example, it's possible to determine the electro-magnetic field of optical structure, as used in telecommunication, solving the Maxwell equations (2^{nd} order PDEs) under suitable conditions (see e.g. [10]). Another application concerns image processing in problems like image smoothing, restoration, segmentation, edge detection and shape analysis. In particular it has been shown how this method can be applied to image multi-scale analysis (see e.g. [5, 2]).

The enhancement of computers performance in these last few years, in terms of calculation speed and available memory, has made it possible to improve the complexity of numerical algorithms for problems requiring huge computational resources. This includes algorithm used to solve PDEs. A numerical program that solves the equation and determines the solution consists generally of the following three parts: a pre-processor, which is the subject of this article, that provides the data structure for the description of the domain; a solver that uses these structures to determine the linear problem that follows from the discretization of the domain; a post-processor that uses the output of the solver for graphical representation.

In Section 2 below, we will show how to divide a three-dimensional domain into tetrahedra. We will show also the algorithm that refines and coarses the grid.

In Section 3 some example of division of the domain will be shown.

2. Discretization of the domain

One important aspect when solving PDEs is the subdivision of the computational domain which is required to obtain an approximate solution that is acceptably close to the exact solution of the differential equation under study. A particularly well known kind of subdivision is the *triangulation*.

A triangulation $\mathcal{T}$ is a set of *non-degenerate* elements with vertices in $\Re^N$, N = 2, 3. The elements of $\mathcal{T}$ are called non-degenerate if the angles of the elements of the triangulation are lower and upper bounded for all $\tau \in \mathcal{T}$. In a three-dimensional domain solid angles will be considered.

$\mathcal{T}$ is called *conforming* if the interSection of any two non disjoint non-identical elements consists either of a common vertex or a common edge or a common face. An element $\tau \in \mathcal{T}$ is said to have a non-conforming node, if there is a node P of the triangulation which is not a vertex of τ but $P \in \mathcal{T}$.

Let's introduce the relation $\mathcal{T}_1 \leq \mathcal{T}_2$ if $\mathcal{T}_2$ is obtained by refinement of $\mathcal{T}_1$. We call a sequence $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2 \ldots$ *stable* if all the elements $\tau \in \cup_\kappa \mathcal{T}_\kappa$ are non- degenerate.

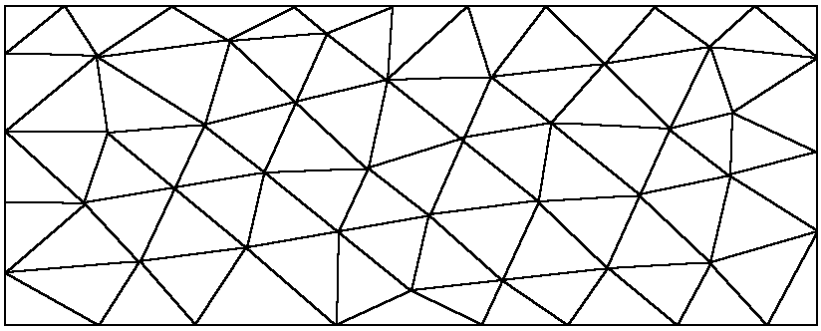


Fig. 1 - Example of a conforming triangulation in $\Re^2$.

The main difficulties to overcome are related to the stability and conformity of the grid. Another important aspect is the *adaptivity* of the triangulation that means that the mesh is refined in regions where high accuracy of the solution is required. The speed of the solver is inversely proportional to the number of elements of the grid, because the number of unknowns increases with the increasing number of elements of the triangulation; therefore, the distribution of elements within the domain is an important aspect.

An element of the domain can be divided using different methods; in this paper the biSection method is used. If we consider, for example, a bidimensional domain, than every element of the grid is a triangle. Each triangle has an edge called *refinement edge* and the subdivision of the triangle will be obtained simply by joining the middle point of the chosen edge with the opposite vertex. It's possible to prove that this method produces a conforming grid in a finite number of division steps. A good choice for the refinement edge would be the longest edge of each triangle, see [3].

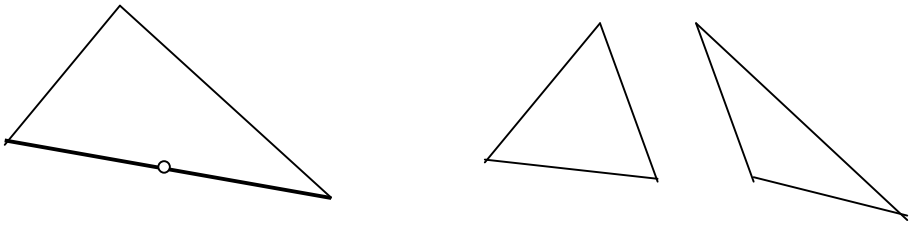


Fig.2 - BiSection of one element in $\Re^2$.

The domain can be divided using two approaches.

The first is a non recursive approach; here, at every step of the refinement algorithm, only one element at a time is divided. This means that after the execution of the refinement algorithm some non conforming points are introduced and that conformity is broken. It is thus necessary to check the grid to find out and bisect all the non conforming elements. The refinement algorithm is the repeated for all the non-conforming elements of the grid until a conforming grid is obtained. Furthermore, this algorithm processes only one element at a time.

The second is a recursive approach; in this case we want to apply the refinement algorithm at most once to an element because we don't want to introduce any non-conforming point after the division process. This is possible if we divide a whole set of elements that share the same refinement edge. It's so clear that in this case, at the end of the refinement algorithm, the conformity of the grid is maintained. This guarantees that the conformity is not lost.

2.1 Three-dimensional case

The starting (macro) grid is one of the fundamental problems to face when working with three-dimensional domains, because the stability of the triangulation depends on it.

Let's consider a cube denoted by M. The starting grid consists of six congruent tetrahedra (see Fig. 3). Three edges of each tetrahedron correspond to edges of M, two are diagonal faces of M and one is the diagonal of M.

This initial embedding was introduced by Bänsch [4] for a non-recursive domain discretization and is excellent for the generation of stable triangulation. Kossaczky [8] has also applied this starting grid and provided a recursive domain discretization algorithm.

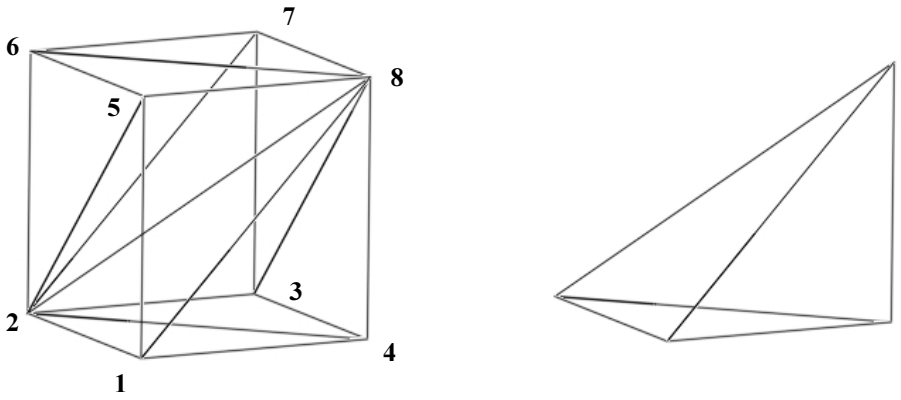
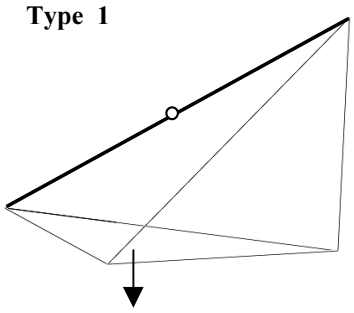


Fig. 3

Let us consider now one of the tetrahedra and divide it by using the biSection method in three successive stages, as shown in Fig. 4. The tetrahedron is divided by cutting the two faces that share the refinement edge. We will consider the longest edge as refinement edge. At every stage of the biSection, two congruent tetrahedra are obtained; in Fig. 4 only one of the two elements that are obtained at each stage is shown. Note that at the end of the three stages, the resulting tetrahedron is congruent to the initial tetrahedron considered before applying the biSection method. This turns out to be very important because we can find out that only three classes of similarity are allowed and hence the (solid) angles of the tetrahedra in the grid are fixed a priori. The three classes of similarity are called type one, type two and type three respectively.



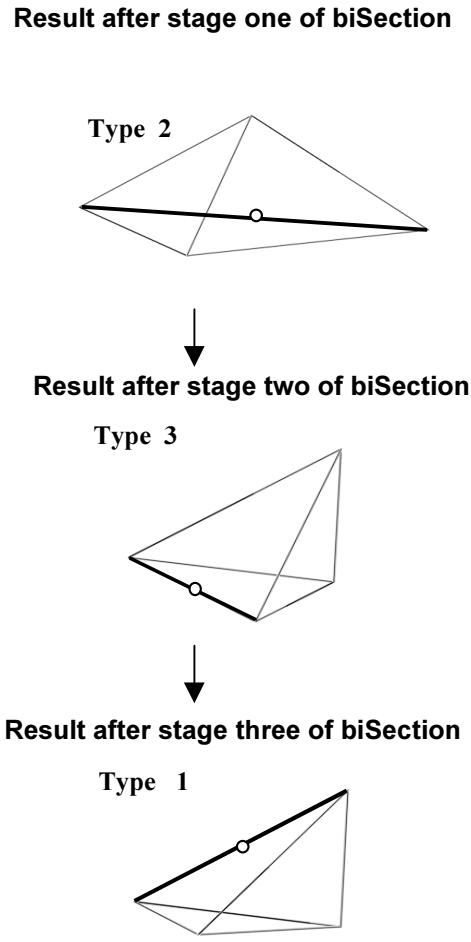


Fig. 4

Let us now consider how to store the data of each tetrahedron. The pre-processor has to create and describe the triangulation with data structures that are the input for the solver. First of all, it is necessary to trace the vertices that forms the tetrahedra, for example V_1, V_2, V_3 and V_4. The best choice for simplifying the biSection algorithm is to store the four vertices related to each tetrahedra in a way that the vertices related to the longest edge are stored in the first two positions. When splitting this tetrahedron, two similar tetrahedra are obtained and it is possible to store the vertices of the two resulting tetrahedra in the same way. This algorithm can easily be implemented as shown in Fig.3, where it is possible to see how to obtain the vertices of two children starting from the father. The only exception is when the division of a type one tetrahedron is considered; in this case a permutation of the elements in the second and third position of the right child's vertices is needed.

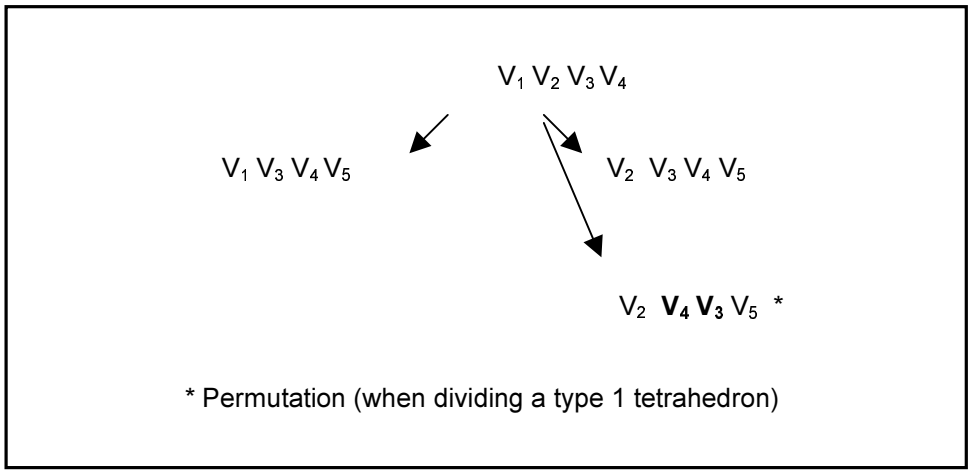


Fig. 5

Another important aspect regards the memory storage for the *neighbors.* That is, for each tetrahedron we have to determine which are the adjacent tetrahedra. Usually, with the expression *neighbors* we are describing those tetrahedra that share a face with the tetrahedron under consideration. In this work we will use a different approach and consider as *neighbors* all the elements that share a common edge. On the one hand this choice increases the size of the data structure, because every tetrahedron has four faces and six edges, but on the other hand it's simpler to find the tetrahedra that share the same refinement edge. It's also possible to easily delete some of the tetrahedra of the grid, for example when studying irregular domains. This aspect will be developed later in the examples Section.

In Fig. 6 the data structure of the tetrahedra and the corresponding neighbors are shown. Every tetrahedron of the grid is numbered, so we can refer to the data of tetrahedron number 'i' simply checking the column 'i' of the related structure. In each column of the first matrix the four vertices of each tetrahedron are stored, and in the second matrix the neighbors of the six edges of each element are stored. Let's now consider two vertices of one of any tetrahedra; by observing the neighbors matrix it is possible to establish which tetrahedron shares the same edge (two vertices) with the first considered element. In fact, it suffices to observe the number of this tetrahedron from the adjacent matrix, in the position corresponding to the pair of vertices chosen. It's possible to repeat the same procedure for the found tetrahedra, thus finding new elements that share this edge. That leads to the concept of *chain.* Note that the order of the tetrahedra inside the chain is not important so it is sufficient for all the elements with two common vertices to be stored in a random order. Of course, the chain has to be closed that means that must be possible to go trough the chain and come back to the element we started.

Matrix of tetrahedra

(referred to fig. 3)

	(1)	(2)	(3)	(4)	(5)	(6)
α	**1**	**1**	**1**	**1**	**1**	**1**
β	**7**	**7**	**7**	**7**	**7**	**7**
γ	**8**	**6**	**6**	**3**	**3**	**8**
δ	**5**	**5**	**2**	**2**	**4**	**4**

Matrix of adjacent

(referred to fig. 3)

α–β	**2**	**3**	**4**	**5**	**6**	**1**
α–γ	**6**	**3**	**2**	**5**	**4**	**1**
α–δ	**2**	**1**	**4**	**3**	**6**	**5**
β–γ	**6**	**3**	**2**	**5**	**4**	**1**
β–δ	**2**	**1**	**4**	**3**	**6**	**5**
γ–δ	**1**	**2**	**3**	**4**	**5**	**6**

Fig. 6

For Example, if the vertices considered are 1 and 7, and we start from tetrahedron 1, the chain will be:

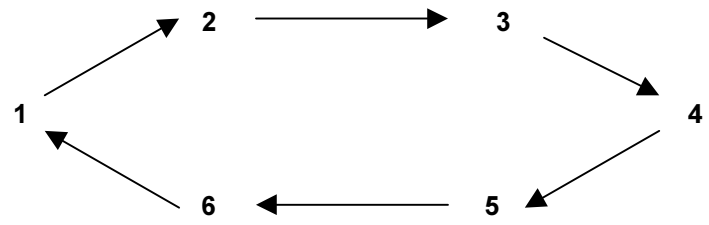


Fig. 7 - Example of chain with vertices 1 and 7.

2.2. Refinement algorithm

This algorithm works for every element of the domain and the refinement edge can also belong to the boundary of the domain.

The refinement algorithm can be implemented as shown below.

Recursive Procedure Refine(*t*)

Let *e* be the *refinement edge* of tetrahedron *t*
Let *d* be the tetrahedron that shares the edge *e* with the tetrahedron *t* (*d* is find out from the adjacent matrix)
while [$d \neq t$] {
    while [*e* is not the *refinement edge* of *d*] {
        Refine(*d*)
    }
    $d' = d$;
    d = tetrahedron that shares the edge *e* with the tetrahedron *d'* (*d* is find out from the adjacent matrix)
}
Refinement of all tetrahedra that share the *refinement edge e*

}

2.3. Coarsen algorithm

This algorithm reverses the refinement process. In this case the *coarse edge* is introduced, which consists of the vertices in position one and four in the “matrix of nodes”. Here, we have to find all the elements that share the same coarse edge and then join every element with the respective son. Note that after the execution of this algorithm, the node in the fourth position of the matrix of nodes does not exist anymore. Thus, this node is removed so the original edge is restored.

The coarsen algorithm can be implemented as shown below.

Recursive Procedure Coarse(t)

Let *e* be the *coarse edge* of tetrahedron *t*
Let *d* be the tetrahedron that shares the edge *e* with the tetrahedron *t* (*d* is find out from the adjacent matrix)
while [$d \neq t$] {
    while [*e* is not the *coarse edge* of *d*] {
        Coarse(*d*)
    }
    d' = *d*;
    d = tetrahedron that shares the edge *e* with the tetrahedron *d'* (*d* is find out from the adjacent matrix)
}
Fusion of all tetrahedra that share the *refinement edge e* with their sons

}

3. Examples

In this Section some examples of adaptive refinement of a domain will be shown.

In Fig. 8 the meaning of adaptivity is explained. Let's suppose that it is important to refine the grid around a certain point called 'A'. It is possible to observe that the size of the tetrahedra around this point are smaller than the ones of the tetrahedra far away from the point considered.

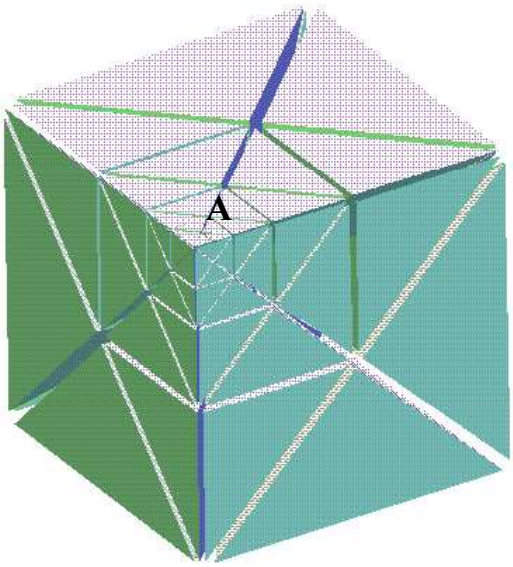


Fig. 8

It is possible to analyze also irregular domains. In this case, as mentioned in Section 2, it is possible to erase some of the tetrahedra of the grid according to an input function. All the tetrahedra that do not match this function are deleted from the matrix of tetrahedra. Of course it is important to update not only the matrix of tetrahedra but also the matrix of adjacents. This is provided by shortening the 'chains' of the neighbors that contain the deleted elements. In Fig. 9 the following domain is considered:

$$\Omega := \{(x,y,z) \mid 0 \le x \le \frac{2}{5} \text{ or } \frac{3}{5} \le x \le 1,\ 0 \le y \le \frac{2}{5} \text{ or } \frac{3}{5} \le y \le 1,\ \frac{2}{5} \le z \le \frac{3}{5}\}$$

It is now possible to study in detail this domain in a well determined zone; we want, for example, to focus the attention inside our domain at the intersection with a sphere.

The result is shown in the picture below.

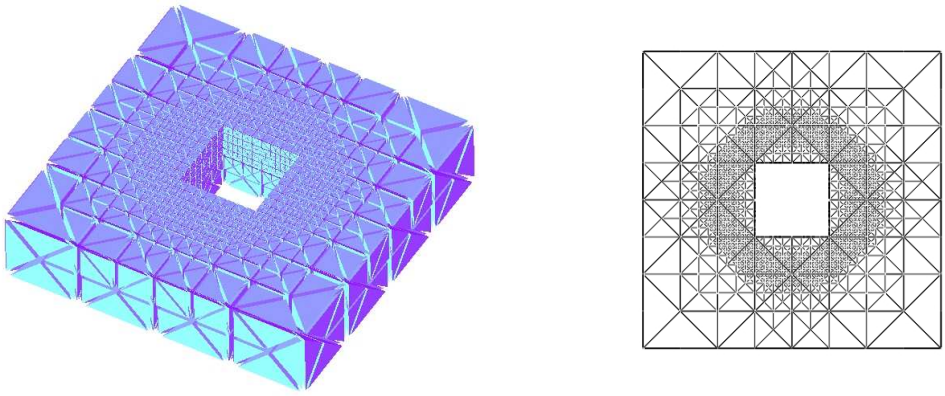


Fig. 9 –Normal view and top view.

References

[1] T. J. Baker, *Developments and trends in three-dimensional mesh generation*, Appl. Numer. Math., 5 (1989), pp.275-307

[2] E.Bänsch, *An adaptive finite-element strategy for the three-dimensional time-dependent Navier-Stokes equations*, J. Comput. Appl. Math. 36 (1991) 3-28

[3] E.Bänsch, *Local mesh refinement in 2 and 3 dimensions*, IMPACT of computing in science and engineering, 3, 181-191 (1991)

[4] E. Bänsch, *Mesh refinement in two and three dimensions*, INRIA (9/95) pp. 307-331, course “Calcul d’erreur a posteriori et adaption de maillage”, Rocquencourt 9/1995

[5] E. Bänsch, K. Mikula, *A coarsening element strategy in image selective smoothing*, computing and visualization science, 53-61 (1997)

[6] D. J. Hebert, Symbolic local refinement of tetrahedral grids, Technical Report ICMA-93-181, University of Pittsburgh, J. Symbolic Comput., 17 (1994), pp. 269-288

[7] D. Hempel, *Local mesh adaptation in two space dimensions*, IMPACT of Computing science end Engineering, 5, no. 4, 309-317 (1993)

[8] I. Kossaczky, *A recursive approach to local mesh refinement in two and three dimensions*, Journal of computational and applied mathematics 55 (1994) 275-288

[9] J. M. Maubach, *Local bisection refinement for n-simplicial grids generated by reflection*, SIAM J. Sci. Comput., 210-227 (1995)

[10] A. Petrini, *Studio di guide ottiche con il metodo degli elementi finiti* , Tesi di laurea, University of Bologna (1996)

[11] M.-C. Rivara, *Selective refinement/derefinement algorithms for sequences of nested triangulation*, International J. Numer. Methods Engrg. 28 (1989) 2889 - 2906

Berichte aus der Technomathematik ISSN 1435-7968

http://www.math.uni-bremen.de/zetem/berichte.html
- Vertrieb durch den Autor –

Reports **Stand: 23. Februar 2000**

98-01. Peter Benner, Heike Faßbender:
An Implicitly Restarted Symplectic Lanczos Method for the Symplectic Eigenvalue Problem, Juli 1998.

98-02. Heike Faßbender:
Sliding Window Schemes for Discrete Least-Squares Approximation by Trigonometric Polynomials, Juli 1998.

98-03. Peter Benner, Maribel Castillo, Enrique S. Quintana Ortí:
Parallel Partial Stabilizing Algorithms for Large Linear Control Systems, Juli 1998.

98-04. Peter Benner:
Computational Methods for Linear-Quadratic Optimization, August 1998.

98-05. Peter Benner, Ralph Byers, Enrique S. Quintana Ortí, Gregorio Quintana Ortí:
Solving Algebraic Riccati Equations on Parallel Computers Using Newton's Method with Exact Line Search, August 1998.

98-06. Lars Grüne, Fabian Wirth:
On the rate of convergence of infinite horizon discounted optimal value functions, November 1998.

98-07. Peter Benner, Volker Mehrmann, Hongguo Xu:
A Note on the Numerical Solution of Complex Hamiltonian and Skew-Hamiltonian Eigenvalue Problems, November 1998.

98-08. Eberhard Bänsch, Burkhard Höhn:
Numerical simulation of a silicon floating zone with a free capillary surface, Dezember 1998.

99-01. Heike Faßbender:
The Parameterized SR Algorithm for Symplectic (Butterfly) Matrices, Februar 1999.

99-02. Heike Faßbender:
Error Analysis of the symplectic Lanczos Method for the symplectic Eigenvalue Problem, März 1999.

99-03. Eberhard Bänsch, Alfred Schmidt:
Simulation of dendritic crystal growth with thermal convection, März 1999.

99-04. Eberhard Bänsch:
Finite element discretization of the Navier-Stokes equations with a free capillary surface, März 1999.

99-05. Peter Benner:
Mathematik in der Berufspraxis, Juli 1999.

99-06. Andrew D.B. Paice, Fabian R. Wirth:
Robustness of nonlinear systems and their domains of attraction, August 1999.

99-07. Peter Benner, Enrique S. Quintana Ortí, Gregorio Quintana Ortí:
Balanced Truncation Model Reduction of Large-Scale Dense Systems on Parallel Computers, September 1999.

99-08. Ronald Stöver:
Collocation methods for solving linear differential-algebraic boundary value problems, September 1999.

99-09. Huseyin Akcay:
Modelling with Orthonormal Basis Functions, September 1999.

99-10. Heike Faßbender, D. Steven Mackey, Niloufer Mackey:
Hamilton and Jacobi come full circle: Jacobi algorithms for structured Hamiltonian eigenproblems, Oktober 1999.

99-11. Peter Benner, Vincente Hernández, Antonio Pastor:
On the Kleinman Iteration for Nonstabilizable System, Oktober 1999.

99-12. Peter Benner, Heike Faßbender:
A Hybrid Method for the Numerical Solution of Discrete-Time Algebraic Riccati Equations, November 1999.

99-13. Peter Benner, Enrique S. Quintana Ortí, Gregorio Quintana Ortí:
Numerical Solution of Schur Stable Linear Matrix Equations on Multicomputers, November 1999.

99-14. Eberhard Bänsch, Karol Mikula:
Adaptivity in 3D Image Processing, Dezember 1999.

00-01. Peter Benner, Volker Mehrmann, Hongguo Xu:
Perturbation Analysis for the Eigenvalue Problem of a Formal Product of Matrices, Januar 2000.

00-02. Ziping Huang:
Finite Element Method for Mixed Problems with Penalty, Januar 2000.

00-03. Gianfrancesco Martinico:
Recursive mesh refinement in 3D, Februar 2000.