Die approbierte Originalversion dieser Dissertation ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

DISSERTATION

Efficient Solution of Large Linear Systems Arising in the 3-Dimensional Modelling of an Electric Field

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors
der technischen Wissenschaften unter der Leitung von

A.Univ.Prof. Dr. Winfried Auzinger
Institut für Angewandte und Numerische Mathematik (E115)

sowie A.Univ.Prof. DDr. Frank Rattay und Dr. Martin Reichel
Institut für Analysis und Technische Mathematik (TU Wien) und
Institut für Biomedizinische Technik und Physik (Universität Wien)

eingereicht an der Technischen Universität Wien
Fakultät für Technische Naturwissenschaften und Informatik

von

Dipl.-Ing. Christoph L. Fabianek
Matrikelnummer: 9725236
Mayerhofgasse 3/407
A-1040 Wien

Wien, im Dezember 2003

Abstract

Functional electrical stimulation (FES) of denervated skeletal muscles found in resent years an entry into rehabilitation of paraplegics. The aim of current research is the optimization of stimulation parameters as well as studying the ramifications of stimulating the muscles in the thigh.

To simulate the distribution of the electric field a 3-dimensional model of the thigh is created which is based on the theory of activation functions from Rattay 1990 and the muscle model from Reichel 1999. With this model it is possible to simulate the electrical activity of muscle fibers. In the current implementation conductivity values based on grey values obtained from CT data are used in the Poisson equation. Via discretization by the method of finite differences and solution of the arising systems this leads to the voltage respectively current distribution. This thesis aims to implement and integrate an efficient solver for the large linear systems arising and shorten the time of the solution process.

After a thorough analysis of the numerical properties of these systems, various solution strategies have been investigated. Beginning with direct solvers which take the symmetric and sparse nature into account, iterative solvers and finally Krylov subspace methods have been implemented and tested. On the later we focused on the preconditioned Conjugate Gradient method.

Because of the bad condition of the given systems the solvers mentioned above led to unsatisfactory results and special preconditioning methods became mandatory. Again various methods have been implemented and tested until multigrid methods finally led to outstanding results in terms of convergence speed. However, the quite high memory requirements could not be satisfied on the target computer and therefore tools for remote computing were used to leverage external machines. Now these multigrid methods are performed remotely on high-end computers and the solution time from previously 9 hours shortens to about an hour.

In the course of the evaluation of the methods described above it became necessary to work with various model problems, and a new simple and especially small thigh model was developed. Using this model existing results could be verified and new insight was gained. In particular, this allows to evaluate new methods easier and quicker than before.

Kurzfassung

Die Funktionelle Elektrostimulation (FES) von denervierter Skelettmuskulatur fand in den letzten Jahrzehnten Eingang in die Rehabilitation Querschnittgelähmter. Ziel der aktuellen Forschung ist es die Optimierung der Simulationsparameter sowie die Auswirkungen auf die Muskulatur des Oberschenkels zu studieren.

Zur Simulation der Potentialverteilung wird ein 3D-Modell des Oberschenkels erstellt, welches auf der Theorie der Aktivierungsfunktionen nach Rattay 1990 und dem Muskelmodell nach Reichel 1999 basiert. Damit ist es möglich die elektrische Aktivität von Muskelfasern zu simulieren. Im konkreten Fall werden dabei aus den CT-Daten anhand der Graustufen die Leitfähigkeitswerte ermittelt und mit Hilfe der Poisson Gleichung, derzeit diskretisiert mit der Methode der Finiten Differenzen, ergibt sich daraus die Strom- bzw. Spannungsverteillung. Ziel der vorliegenden Arbeit ist die effiziente Lösung der dabei auftretenden linearen Gleichungssysteme und die Implementation und Integration eines Gleichungslösers.

Nach einer Analyse der numerischen Eigenschaften der auftretenden Systeme wurden verschiedene Lösungsmethoden getestet. Ausgehend von direkten Lösern, die die Symmetrie und die schwache Besetztheit ausnutzen, wurden iterative Verfahren und schließlich Krylov Subspace Verfahren, insbesondere das vorkonditionierte Konjugierte Gradientenverfahren, implementiert und getestet.

Aufgrund der schlechten Konditionierung der Gleichungssysteme lieferten die erwähnten Lösungsmethoden nur unbefriedigende Ergebnisse und es erwies sich eine Vorkonditionierung als unbedingt notwending. Auch hier wurden verschiedene Methoden entwickelt und erprobt, bis schließlich Mehrgitterverfahren hervorragende Ergebnisse bezüglich der Laufzeit lieferten. Allerdings stellt dieses Verfahren hohe Anforderungen an die Systemressourcen des verwendeten Computers. Daher erwies es sich als notwendig Software für verteiltes Rechnen zu verwenden, um auf externe Hochleistungscomputer zuzugreifen, auf denen diese Mehrgitterverfahren ausgeführt werden. Damit ist es nun möglich die bisherige Lösungszeit von 9 Stunden für eine Simulation auf etwa eine Stunde zu verringern.

Im Zuge der Bewertung der oben beschriebenen Verfahren war es notwendig mit verschiedenen Modellproblemen zu arbeiten. Insbesondere wurde ein einfaches (und vorallem viel kleineres) Modellproblem für den Oberschenkel entwickelt. Anhand dessen war es möglich, vorhandene Ergebnisse zu bestätigen und neue Erkenntnisse zu gewinnen. Damit steht jetzt auch ein Werkzeug zur Verfügung, anhand dessen neue Verfahren einfacher und schneller getestet werden können.

Contents

1 Introduction 1

2 Medical and Physical Basics 3

2.1 Medical Background 3

2.1.1 Human Thigh 3

2.1.2 Electrical Stimulation of Nerve and Muscle Fibers 4

2.2 Physical Interpretation 7

2.3 Thigh Model 9

3 Mathematical and Numerical Investigation 13

3.1 Discretization of the Poisson Equation 13

3.1.1 Finite Difference Method 13

3.1.2 Finite Element Method 16

3.2 Matrix Properties 20

4 Solution Methods 24

4.1 Sparse Direct Solvers 24

4.2 Basic Iterative Methods 26

4.3 Projection Methods 28

4.4 Krylov Subspace Methods 29

4.4.1 GMRES algorithm 30

4.4.2 Conjugate Gradient algorithm 32

4.4.3 Overview 35

5 Preconditioning 37

5.1 Preconditioned Iterations 37

5.2 Cholesky Factorization 39

5.3 Multigrid Methods 41

5.3.1 Two-grid cycle 42

5.3.2 Components of Multigrid 43

5.3.3 Multigrid and Full Multigrid Cycle 45
5.3.4 Algebraic Multigrid 49
5.3.5 Multigrid as preconditioner 50

6 Implementation 52
6.1 PEBBLES 52
6.1.1 Field of Application 52
6.1.2 Porting PEBBLES to Other Architectures 53
6.2 NetSolve 54
6.2.1 The NetSolve System 55
6.2.2 Interfaces 56
6.2.3 Load Balancing 57
6.2.4 Fault Tolerance 58
6.2.5 Integration of PEBBLES into NetSolve Computational Servers 58
6.3 HARNESS 60
6.3.1 Architecture 60
6.3.2 Including PEBBLES into HARNESS 62
6.4 Interface to the FES-Tool (`solver.m`) 62

7 Numerical Experiments 63
7.1 Test Environment 63
7.2 Data Set Analysis 71
7.3 Parameter Analysis 73
7.4 Numerical Performance 77
7.5 Remote Computing 79

8 Conclusions 82

9 Outlook 84

A Solver Configuration 85

B Curriculum Vitae 87

Bibliography 93

List of Algorithms

1 A simple sparse Gaussian elimination. 25
2 General Jacobi Iteration. 26
3 General Gauss-Seidel Iteration. 26
4 General SOR Iteration. 27
5 Steepest Descent Iteration. 28
6 Arnoldi-Modified Gram-Schmidt. 30
7 General Minimum Residual (GMRES) algorithm with restarting after n steps. 32
8 Conjugate Gradient algorithm. 35
9 Preconditioned Conjugate Gradient algorithm. 39
10 Block Tri-Diagonal Cholesky Factorization. 39
11 Two-grid cycle. 43

Acknowledgements

First I would like to express my gratitude to my advisor Winfried Auzinger for his patience and the granted freedom at this doctoral thesis which is necessary for an autonomous work. I also want to thank Winfried Mayr for his short-dated acceptance as examiner for this work.

The brain-modelling group led by Frank Rattay introduced me into the exciting world of neuroscience and was always a welcome place for fruitful discussions. Especially Martin Reichel at the Department of Biomedical Engineering and Physics was a valuable source for all the questions that arose in the field of functional electrical stimulation. Many thanks to them all.

During the work on this thesis Jack Dongarra gave me the unique opportunity to visit his Innovative Computing Laboratory at the University of Tennessee in Knoxville for 5 months. At this stay my work experienced a vital impact in the use of methods for distributed computing. My sincere thanks to him as well as to Don Fike and Keith Seymour from the lab who were great friends and introduced me to the southern lifestyle.

Many thanks also to my colleagues at university and dormitory, especially to Wilfried Gansterer, Christa Kreuzmayr, Johannes Martinek, Michael Schneider and Alois Zoitl for all their support in technical and non-technical matters during my studies.

Der größte Dank aber geht an meine Eltern und Geschwister für die Unterstützung bei meinem Studium.

CHRISTOPH FABIANEK

Chapter 1

Introduction

Functional electrical stimulation (FES) is the clinical application of a small electric current to the intact nerves or denervated skeletal muscles, in order to elicit a muscle contraction. This contraction is then incorporated into a functional activity, for example standing up or eventually walking. Therapeutic stimulation is aimed to improve quality of life for individuals with disabilities e. g., stroke victims, multiple sclerosis sufferers, or patients with spinal cord injuries.

In the second half of the 20th century substantial progress was made in this field and a series of electrical stimulators were developed and found their application in several clinical areas. Outstanding examples are the artificial cardiac pacemaker in 1952 (Paul M. Zoll), phrenic pacemakers against respiratory insufficiency in 1966, auditory prothesis (cochlea-implant) for the deaf in 1970, motor nerve stimulation for the paralyzed—leg pacemaker in 1973 and hand pacemaker in 1988 (Rattay [42]). Clinical successes in stimulation of denervated muscles are dated only a few years back by Kern [31] in 1995. So the knowledge was mainly the product of years of experimental use and was therefore mostly empirical and subjective.

In the late 1990's a research group at the Vienna University Clinic (University of Vienna, Department of Biomedical Engineering and Physics at the Vienna General Hospital) started to develop a 2-dimensional model of denervated skeletal muscles (Reichel [44]) and simulated the effects of FES in the thigh. Based on this work a 3-dimensional model (Breyer [11]) was created, anisotropy taken into account (Grotz [25]) and a tool for analyzing and visualizing the areas of activation (Martinek [38]) was implemented. The difficulty in all the models and tools was the solution of the large linear systems arising from the discretization of the problem on calculation of the voltage / current distribution within the thigh. The efficient solution of these systems is subject of this PhD thesis.

Chapter 2 gives an introduction into the medical background and the underlying physical model followed by an analysis of the mathematical properties of the system (Chapter 3).

In the next two chapters (4 and 5) we test the given problem against various solution strategies from direct solvers to Krylov subspace methods and focus on preconditioning techniques. This leads us finally to multigrid methods which are considered as being the fastest numerical methods for the solution of discretized elliptic partial differential equations with which we deal here.

Afterwards we turn to implementation issues of the solver in Chapter 6 and also describe two tools for remote computing to leverage external computing resources. This became necessary because memory requirements for the used solver could not be satisfied at the 32bit computer on which the FES-Tool is used.

Chapter 7 describes in detail the tests to evaluate the discussed algorithms and their results. Based on this information our conclusion (Chapter 8) gives recommendations how to optimally use the developed solvers. The thesis ends with an outlook to further fields of investigation.

Chapter 2

Medical and Physical Basics

This chapter describes the medical background of the relevant physiological principles at the stimulated regions. It continues with the nerve model of Rattay, which explains the activation function and the stimulation of denervated skeletal muscles. Afterwards the physical / mathematical model is outlined which is used to calculate the electrostatic field in the thigh. The chapter concludes with presenting the tools that are used to perform and analyze an FES simulation at the Department of Biomedical Engineering and Physics.

2.1 Medical Background

2.1.1 Human Thigh

The most relevant tissues in the human thigh with influence to FES are skeletal muscles, connective tissue, tendons, fat, skin and bones. The muscle tissue is especially interesting for the stimulation. The passive electrical properties of the muscle are together with the surrounding tissue responsible for how the electrical field builds up. On the other hand is the muscle fiber an active electrical element, which generate action potentials upon contraction. The other tissue influences solely the electrical field.

Skeletal muscle (Fig. 2.1) consists of thousands of elongated, cylindrical cells (approximately 5 -200 μm in diameter and up to 30 cm in length), called muscle fibers, arranged parallel to one another. Each muscle fiber is covered by a plasma membrane called the sarcolemma. Its high resistance is responsible for the lower conductivity orthogonal to direction of fibers. Conductivity within a fiber and outside of the membrane is about decouple the conductivity orthogonal to the direction of fiber. This property is called anisotropy and has to be considered in the simulation.

The fascia lata is a deep fascia that encircles the entire thigh. The thigh musculature is separated in three compartments by deep fascia. The anterior muscles of the thigh, which form the anterior compartment, in particular the quadriceps femoris, causes extension of the knee, which is essential for regaining the ability to get up, stand and eventually walk in case of flaccid paraplegia. Thus the FES is applied to the quadriceps, which itself consists of 4 parts and has therefore to

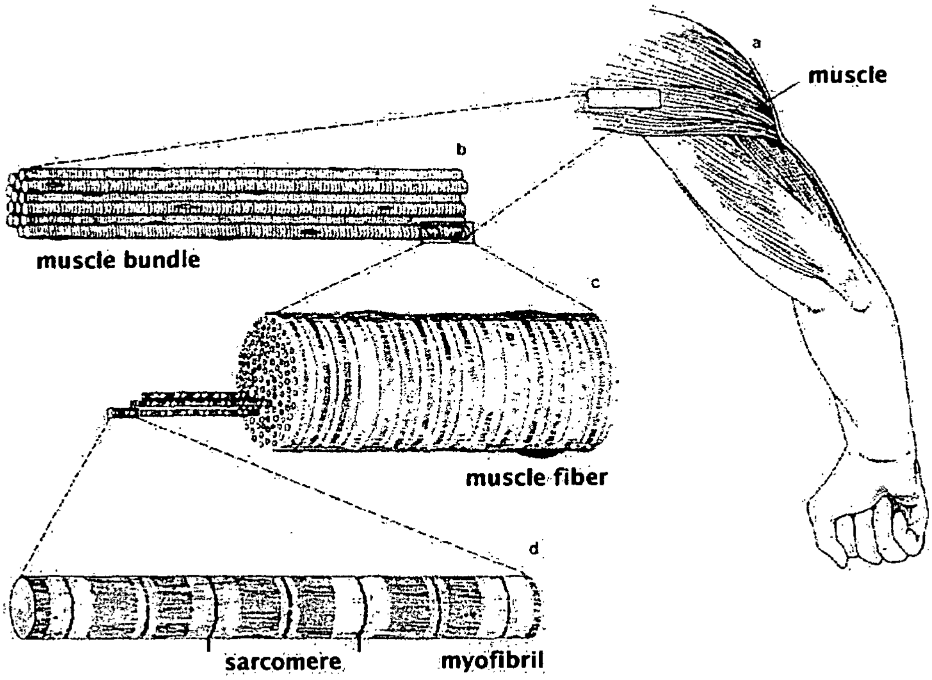


Figure 2.1: Organization of skeletal muscle from gross to molecular levels, after Junqueira et al. [30]

be stimulated with large surface electrodes instead of a single mono-polar point electrode. Fig. 2.2 shows a 3-dimensional model of the muscles of the thigh.

In case of a lesion of the motoric efferent nerve the propagation of an action potential in the peripheral nervous system is interrupted and motoric units in the muscles cannot be innervated in a physiological way. The next subsection explains the electrical and chemical processes in the muscles and nerves when a contraction occurs and introduces the basis for FES.

2.1.2 Electrical Stimulation of Nerve and Muscle Fibers[1]

As a consequence of different ionic concentrations at the inside and the outside of a nerve or muscle fiber, the interior of the cell is normally maintained at a potential of a about 50-70 mV negative to the exterior. When an action potential is produced the voltage is changed to positive values before it falls back again to the resting state (Fig. 2.3). Such an action potential propagates because it disturbs the resting region ahead by itself and channel activities are evoked there, too. A similar effect can be reached artificially by making the inside potential

---

[1] after Rattay [42]

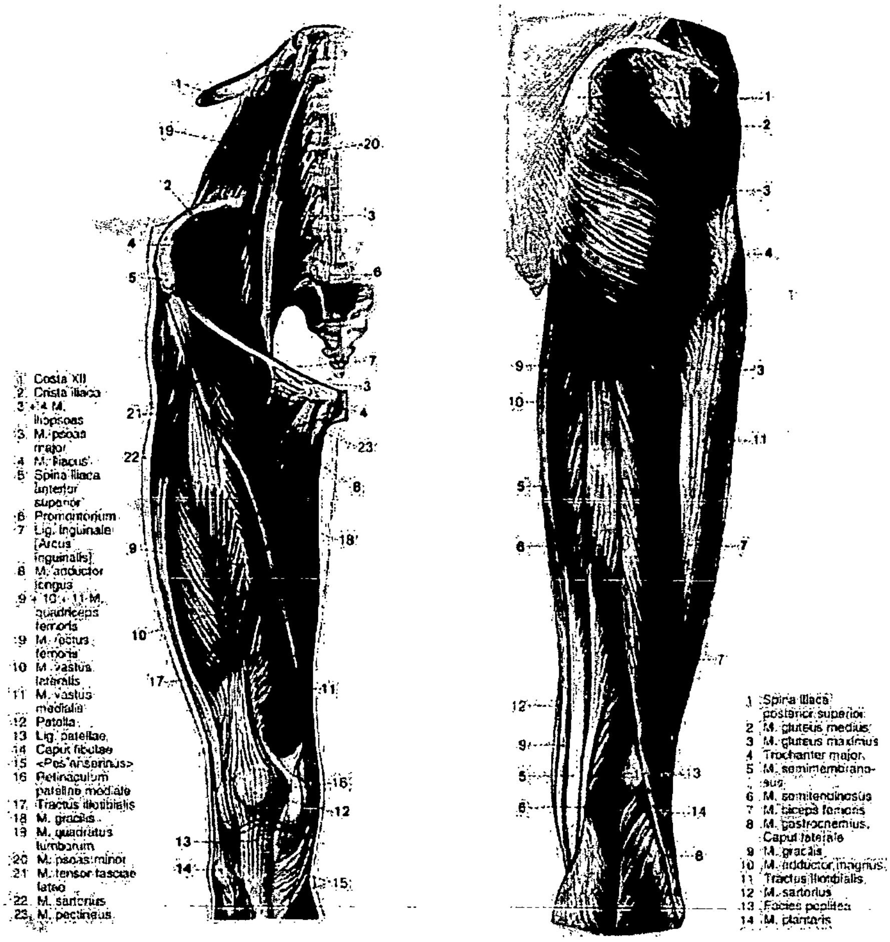


Figure 2.2: Muscles that move the femur (thigh bone), after Lippert [35]

more positive with the help of an inserted micro-electrode. However, placing an electrode inside the cell is usually not practical for clinical applications; therefore, nerve and muscle fibers are usually stimulated by changing the membrane voltage via the extracellular potential.

In order to stimulate nerve fibers artificially an electric field must be created. For this purpose, neural protheses use either surface electrodes or implanted electrodes[2]. Both types of electrodes have their applications.

Surface electrodes need no surgery, but the large distances to the stimulated areas and the insulation of the skin and fat areas demand high stimulation strengths with low fiber selectivity. Implanted electrodes on the other hand demand high

---

[2] An alternative technique is the non-invasive stimulation with coils, see e. g., Carbunaru and Durand [13]

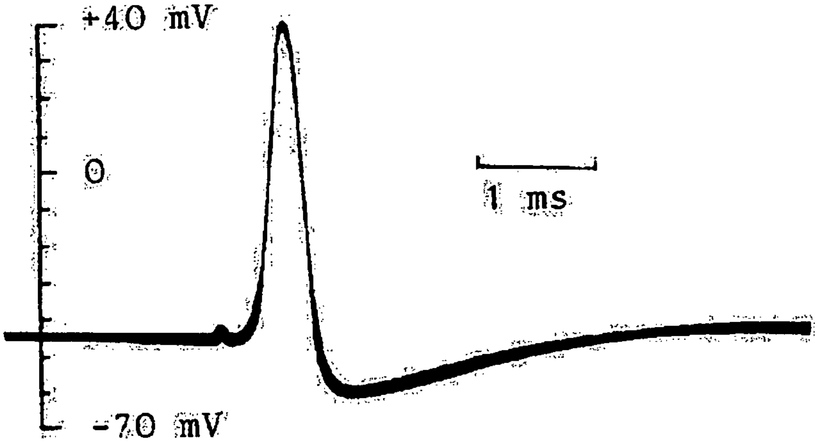


Figure 2.3: An action potential from a squid giant axon. The vertical scale indicates membrane voltage in mV (Hodgkin and Huxley [29]).

quality materials because the surface should not be changed electrolytically. If the electrodes are used for neuromuscular stimulation they consist mostly of very thin coiled stainless steel wires which move with the muscles and other tissues without breakage due to mechanical stress.

For the force control of stimulated muscles the time behavior is of high interest, because the firing patterns of the stimulating nerve or muscle can evoke either smooth or rippled motions. Fig. 2.4 shows the force response of a skeletal muscle, as a function of stimulus frequency. In the figure firing rates below 25–30 Hz produce contractions which are not smooth enough for most practical purposes. Smooth contractions at high force levels are obtained with higher stimulation frequencies, but if the motor nerve is firing with more than 50 Hz the fatigue effect is very fast.

For reasons of safety, biphasic stimulus signals are preferred for most applications since charge accumulation can be avoided. High charge densities can be avoided by using trains of biphasic impulses. The function of the primary pulse is to produce an action potential and the second pulse is used to reverse the electrochemical process. The second pulse also has a hyper-polarizing effect and reduces the stimulating work of the first pulse especially when it is applied shortly after the onset of the first pulse. A delay between these pulses is therefore used because it will reduce this effect. Fig. 2.5 shows such a signal.

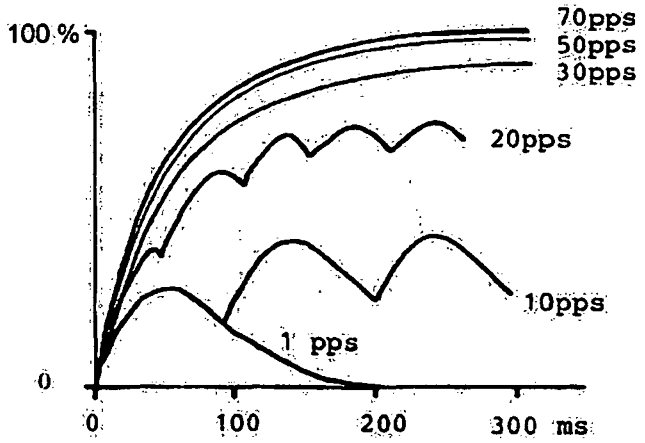


Figure 2.4: Muscle force as a function of stimulation frequency (Solomonow [50]). The force of a skeletal muscle depends on the pulses per second (pps).

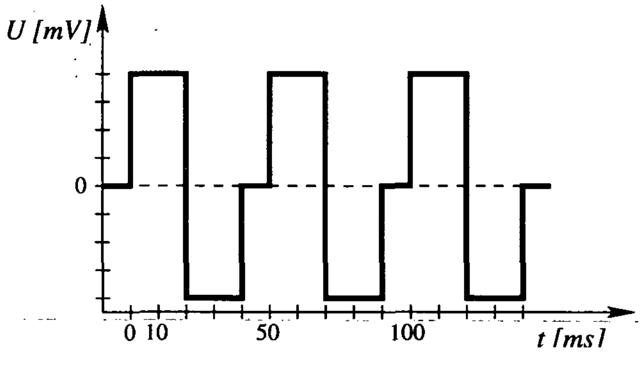


Figure 2.5: Trains of biphasic constant voltage (cv) signals are used.

2.2 Physical Interpretation

When applying voltage or current sources to the human thigh the resulting electrostatic field provides information about triggering an action potential in the muscles. To calculate the electrostatic field the conductivity together with the location of the current or voltage sources is needed.

Today computer tomography is used[3] to produce sliced images of the extremities. Each tissue absorbs the X-rays in a characteristic manner (see Fig. 2.6) and provides the raw data for calculating the conductivity. Through segmentation color values are associated with tissue types which have to be applied with a correction scheme because of sometimes ambiguous color information in crossover sections (Mandl [37]). Because of anisotropy (especially in muscles) the longitudinal and

---

[3] Another possibility would be to use impedance tomography (Li [34]).

transversal conductivity is calculated (Grotz [25]) based on the location of muscle fibers (Martinek [38]). With this data a 3-dimensional matrix can be built up as base for calculating the electrostatic field.

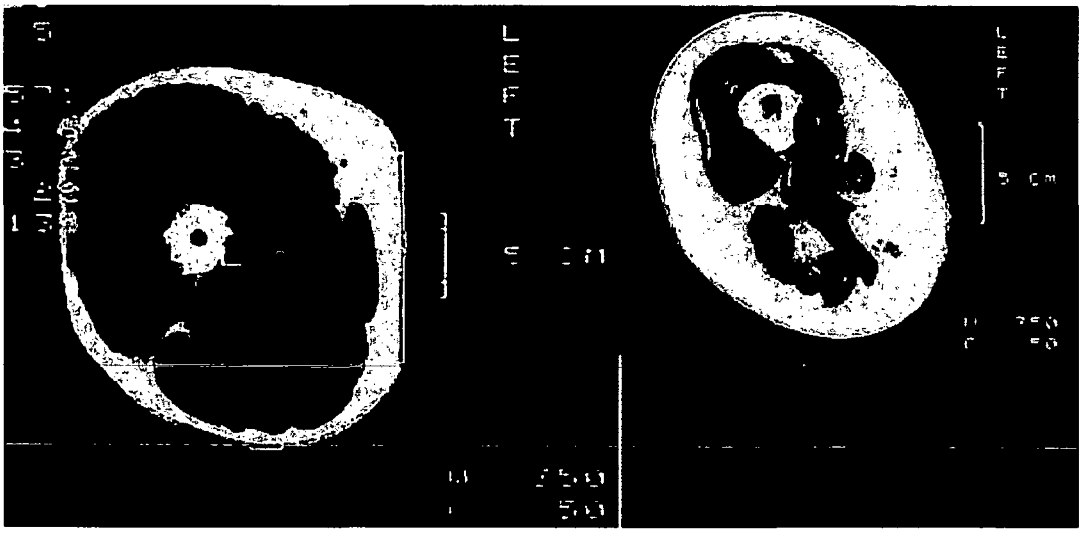


Figure 2.6: Processed computer tomography in the middle part of the left thigh. On the left normal (healthy) and on the right side denervated musculature (from Kern [31]).

The Poisson Equation for electrostatic fields

$$-\nabla\left(G(\boldsymbol{x})\cdot\nabla u(\boldsymbol{x})\right)=0 \quad \text{for } \boldsymbol{x}=(x_1,x_2,x_3) \text{ in } \Omega \tag{2.1}$$

(an elliptic boundary value problem) describes the voltage or current distribution u in a medium with variable conductivity G where Ω is a bounded, open domain in $\mathbb{R}^3$ when no inner sources are present. Equation (2.1) is to be satisfied only for points that are located at the interior of the domain Ω. The conditions on the boundary Γ of Ω are

- Dirichlet boundary conditions

  $$u(\boldsymbol{x})=\varphi(\boldsymbol{x}) \tag{2.2}$$

  for defining the voltage or current sources φ and

- Neumann boundary conditions

  $$\frac{\partial}{\partial n}u(\boldsymbol{x})=G(\boldsymbol{x}) \tag{2.3}$$

  for describing the behavior of the field at the crossover from skin to air (with n denoting the orthogonal vector to the boundary curve). At the hip and knee additionally a damping factor is introduced that simulates some "fade of" of the electric potential contrary to the surrounding air which acts as an insulator.

The conductance G between two voxel for a given direction is based on the specific conductance γ in each voxel that itself is obtained from the colour values of the CT data. To calculate G we us the fact that the elctric resistance R between to voxel is the sum of the resistance in each voxel and that the conductance G is the inverse of R. This leads us to the formula

$$R = R_1 + R_2 = \frac{1}{G_1} + \frac{1}{G_2} = \frac{1}{G} \quad \Rightarrow \quad G = \frac{G_1 \cdot G_2}{G_1 + G_2}.$$

The conductance G in a specific voxel for a given direction is calculated as

$$G = \frac{\gamma A}{l} S,$$

with A the area between two voxel the current has to pass through, l the length from center to the boundary of the voxel and S the factor of anisotropy in the medium for a given direction. E. g., the conductivity G_z in longitudinal direction and discretization with voxel size Δx, Δy, Δz is

$$G_z = \frac{\gamma \, \Delta x \, \Delta y}{\Delta z/2} S_z.$$

At the discretization of the Poisson equation (see Chapter 3) the system of linear equation has to be transformed in the following way depending on the type.

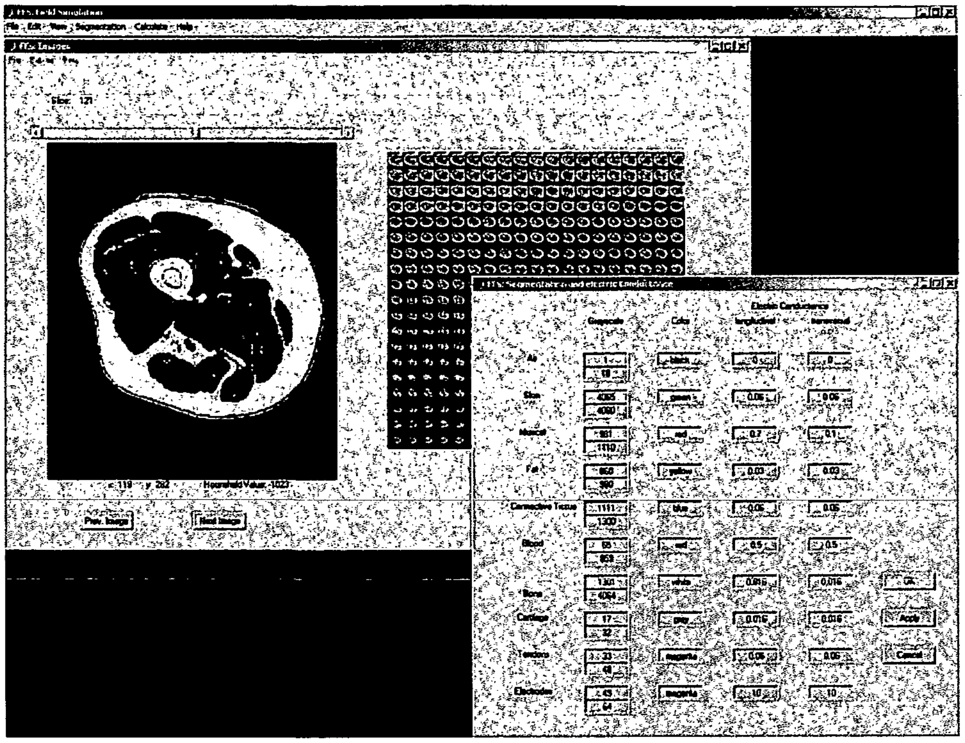
Voltage sources Rows at voltage sources are replaced with 0 except the main diagonal element is set to -1 since at this point the voltage V_0 is given.

Current sources The row / column with the largest entry in the right hand side vector b is removed, because otherwise it would lead to an over-determined system of linear equations.

The right hand side vector b defining the Dirichlet boundary conditions is set to 0 and at the current / voltage sources with 1 respectively -1, which allows an easy scaling of the solution vector to the actual values (usually up to 250 Milliampere respectively ±40 Volt are used).

2.3 Thigh Model

At the Department of Biomedical Engineering and Physics the *FES-Tool* (Functional Electrical Stimulation – Tool) was developed which integrates the above described procedures in an easy to use graphical user interface (Fig. 2.7).



	Grayscale	Color	Electric Conductance longitudinal	 transversal
Air	1 [illegible]	black	0	0
Skin	[illegible] [illegible]	green	0.05	0.05
Muscel	[illegible] 1110	red	0.7	0.1
Fat	[illegible] [illegible]	yellow	0.03	0.03
Connective Tissue	1111 1300	blue	0.05	0.05
Blood	[illegible] [illegible]	red	0.5	0.5
Bone	[illegible] 4064	white	0.016	0.016
Cartilage	17 32	gray	0.016	0.016
Tendons	33 48	magenta	0.05	0.05
Electrodes	[illegible] 64	magenta	10	10

Figure 2.7: Main window of the FES-Tool with a cross-section and segmentation configuration dialog.

In the first step DICOM data as provided by computer tomography is imported. The goals of DICOM are to achieve compatibility and to improve work-flow efficiency between imaging systems and other information systems in health-care environments. This should be established through standards for communication of bio-medical diagnostic and therapeutic information in disciplines that use digital images and associated data.

Next the region and resolution of data to be analyzed is selected (usually the data provided from computer tomography includes both legs and separate calculation for each has to be performed). Thereupon segmentation assigns a conductivity value to each image pixel through association of tissue to color values. After that the position of the electrodes is defined and as result cross sections or a 3D model can be viewed – see Fig. 2.8.

After all data is available the solver is started and the results are displayed as equipotential lines in the computer tomography images (Fig. 2.9).

When the solution is available FES-Analyze (Martinek [38]) finally allows to visualize the areas of activation by calculating the first and second derivative. Fig. 2.10 shows a screenshot of this tool.

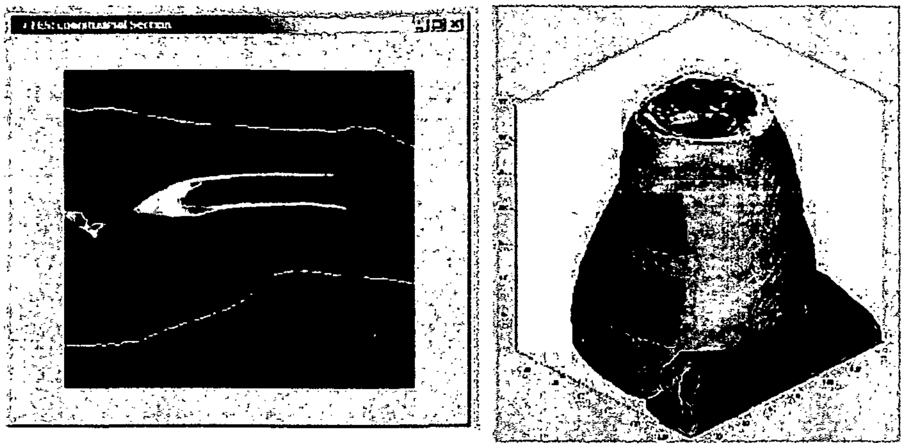


Figure 2.8: 2D and 3D view of the thigh within the FES-Tool.

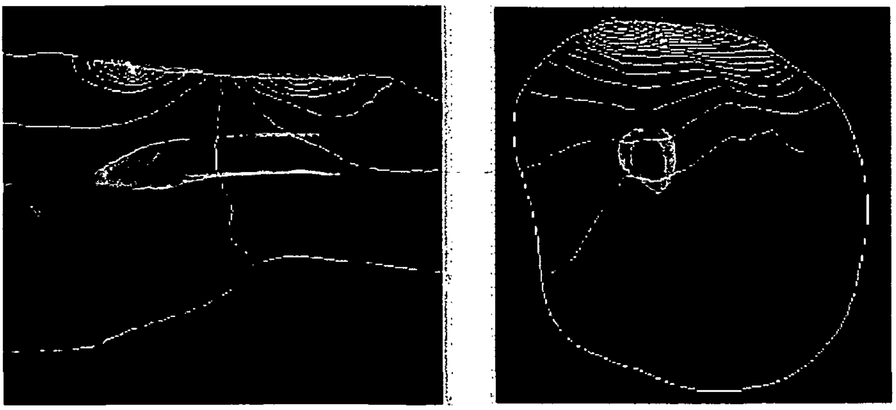
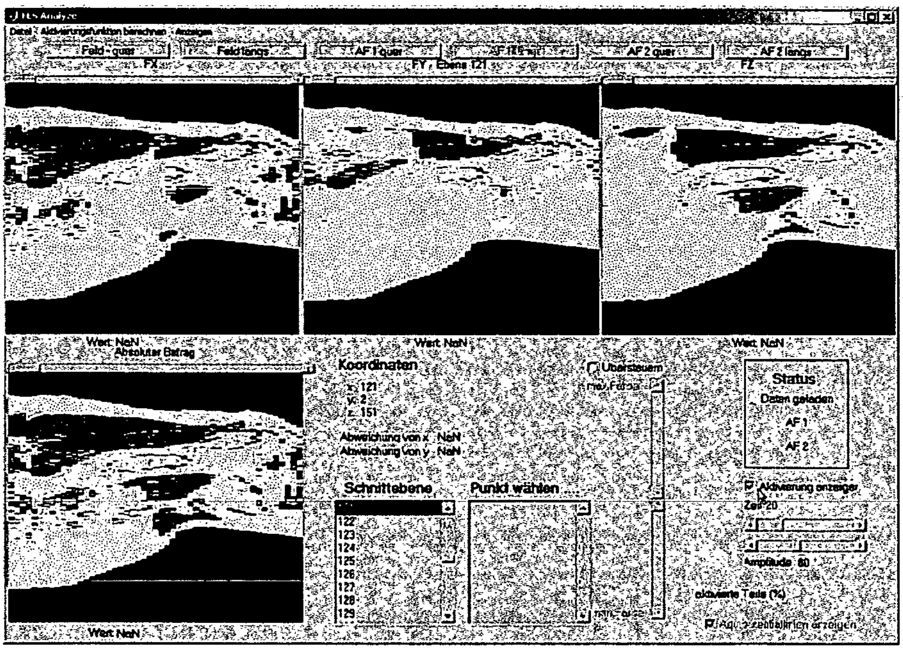


Figure 2.9: The electrical potential in the form of contour plots in the thigh.



Ebene 121

x: 121
y: 2
z: 151

Abweichung von x : NaN
Abweichung von y : NaN

122
123
124
125
126
127
128
129

☑ Aktivierung anzeigen

Zeit 20

Amplitude 80

Figure 2.10: FES-Analyze showing longitudinal cross-sections with the areas of activation in magenta and cyan.

Chapter 3

Mathematical and Numerical Investigation

This chapter introduces the mathematical properties of the systems arising when calculating the current or voltage distribution in the thigh. It describes different ways how to transform the 3D conductance matrix into a system of linear equations, and how to solve such systems. In particular, the Finite Element and the Finite Difference Method are discussed.

Furthermore we give details about the characteristics of the current systems available and relate them to a first discussion of available solution strategies for linear equations.

3.1 Discretization of the Poisson Equation

The common way to solve Partial Differential Equations (PDEs) numerically is to discretize them, i. e., to approximate them by equations that involve a finite number of unknowns. The matrix problem that arise from these discretization is generally large and sparse. There are several different ways to discretize a PDE. The simplest method uses *Finite Difference* approximations for the partial differential operators. The *Finite Element Method* replaces the original function by a function which has some degree of smoothness over the global domain, but which is piecewise polynomial on simple cells, such as small triangles or rectangles. In between these two methods, there are a few conservative schemes called *Finite Volume Methods*, which are designed to emulate continuous conservation laws of physics. This section describes the use of the *Finite Difference* and the *Finite Element Method* in context of the Poisson Equation from Chapter 2.

3.1.1 Finite Difference Method

The *Finite Difference* method is based on local approximations of the partial derivatives in a Partial Differential Equation, which are derived by low order Taylor series expansions. It is particularly appealing for simple regions, such as rectangles, and when uniform meshes are used. The matrices that result from these discretizations are often well structured, which means that they typically consist of a few nonzero diagonals.

To approximate the second derivative of a function u at the point x, the formula

$$\frac{d^2u(x)}{dx^2} = \frac{u(x+h) - 2u(x) - u(x-h)}{h^2} - \frac{h^2}{12}\frac{d^4u(\xi)}{dx^4} \qquad (3.1)$$

is used (for a function u that is C^4 in the neighborhood of x, with h tends to zero and ξ is in the interval $(x-h, x+h)$). This is called the centered difference approximation since the point x at which the derivative is approximated is the center of the points used for the approximation. The dependence of this derivative on the values of u at the points involved in the approximation is often represented by a "stencil" or a "molecule," shown in Fig. 3.1.

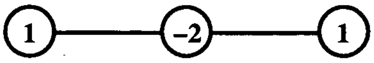


Figure 3.1: The tree-point stencil for the centered difference approximation to the second order derivative.

If the approximation (3.1) is used for three directions in 3D space, the following second order accurate approximation results (in the simplest case, the $\frac{\partial^2}{\partial x_1^2}$, $\frac{\partial^2}{\partial x_2^2}$ and $\frac{\partial^2}{\partial x_3^2}$ terms in the Laplace operator use the same mesh size h for x_1, x_2 and x_3):

$$\begin{aligned}\Delta u(\boldsymbol{x}) \approx \frac{1}{h^2}\,[&u(x_1+h, x_2, x_3) + u(x_1-h, x_2, x_3) + u(x_1, x_2+h, x_3)\\ &u(x_1, x_2-h, x_3) + u(x_1, x_2, x_3+h) + u(x_1, x_2, x_3-h)\\ &-6u(x_1, x_2, x_3)]\,. \qquad (3.2)\end{aligned}$$

This is called the seven-point centered approximation to the Laplacean and the stencil of this finite difference approximation is illustrated in Fig. 3.2.

Consider the three dimensional equation

$$\begin{aligned} -\left(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_x^2} + \frac{\partial^2 u}{\partial x_3^2}\right) &= f \quad \text{in } \Omega \qquad (3.3)\\ u &= 0 \quad \text{on } \Gamma \qquad (3.4)\end{aligned}$$

where Ω is a cuboid $(0, l_1) \times (0, l_2) \times (0, l_3)$ and Γ its boundary. This can be discretized uniformly by $n_i + 2$ points in each direction:

$$x_{i,j} = j \times h_i, \quad j = 0, \ldots, n+1, \quad i = \{1, 2, 3\}$$

where

$$h_i = \frac{l_i}{n_i + 1}.$$

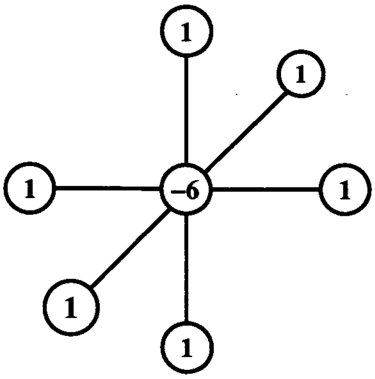


Figure 3.2: The seven-point stencil for the centered difference approximation to the 3D Laplace operator.

Since the values at the boundaries are usually known (Dirichlet boundary conditions), only the interior points are numbered, i. e., the points $(x_{1,k}, x_{2,l}, x_{3,m})$ with $0 < k < n_1$, $0 < l < n_2$ and $0 < m < n_3$. These points are labeled slice-wise, bottom up, one horizontal line at a time. This labeling is called natural ordering (or lexicographical odering) and is displayed in Fig. 3.3 for the simple case $n_1 = n_2 = n_3 = 4$.

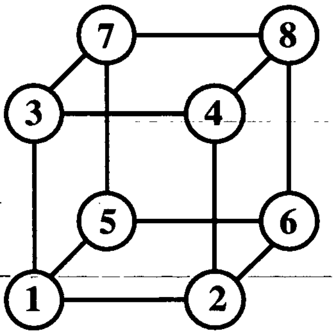


Figure 3.3: Natural ordering of the unknowns for a 4 × 4 × 4 three-dimensional grid. For reasons of a clear illustration border elements (Γ) have been omitted.

If the centered difference approximation is used, then by (3.3) expressed at an interior point $x_{i,j,k}$, the unknowns $u_{i,j,k}, u_{i\pm1,j\pm1,k\pm1}$, satisfy the relation

$$-u_{i-1,j,k} - u_{i,j-1,k} - u_{i,j,k-1} - u_{i+1,j,k} - u_{i,j+1,k} - u_{i,j,k+1} + 6u_{i,j,k} = h^2 f_{i,j,k} \quad (3.5)$$

in which $f_{i,j,k} \equiv f(x_{i,j,k})$. Notice that for $i, j, k = 0$ and $i, j, k = n$ the equation will involve known quantities. Thus, for $n_1 = 5$, $n_2 = 3$ and $n_3 = 4$ the linear system obtained is of the form

$$Ax = f$$

where the pattern of the matrix A appears in Fig. 3.4.

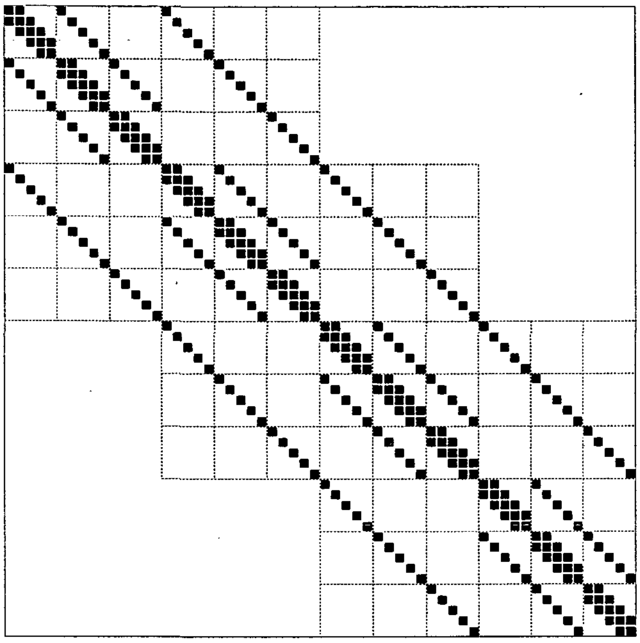


Figure 3.4: Pattern of matrix associated with a 5 × 3 × 4 finite difference mesh.

In "block notation" the matrix has the following block structure:

$$A = \frac{1}{h^2}\begin{pmatrix} B & -I & & \\ -I & B & -I & \\ & -I & B & -I \\ & & -I & B \end{pmatrix} \quad \text{with } B = \begin{pmatrix} C & -I & \\ -I & C & -I \\ & -I & C \end{pmatrix}$$

$$\text{and } C = \begin{pmatrix} 6 & -1 & & & \\ -1 & 6 & -1 & & \\ & -1 & 6 & -1 & \\ & & -1 & 6 & -1 \\ & & & -1 & 6 \end{pmatrix}.$$

3.1.2 Finite Element Method[1]

In the finite difference approximation of a differential equation, derivatives are replaced by difference quotients which involve the values of the solution at discrete mesh points of the domain. The resulting discrete equations are solved, after imposing the boundary conditions, for the values of the solution at the mesh points.

---

[1] See Reddy [43] and Chandrupatla and Ashok [14] for an in-depth discussion of the finite element method.

Although the finite difference method is simple in concept, it suffers from several disadvantages. The most notable are the inaccuracy of the derivatives of the approximated solution, the difficulty in imposing the boundary conditions along non-straight boundaries, the difficulty in accurately representing geometrically complex domains, and the inability to employ nonuniform and non-rectangular meshes.

In the variational solution of differential equations, the differential equation is cast into an equivalent variational form, and then the approximate solution is assumed to be a linear combination ($\sum c_j\phi_j$) of given basis functions ϕ_j. The parameters c_j are determined from the variational form.

The finite element method provides a systematic procedure for the derivation of the approximating functions. The method is endowed with two basic features which account for its superiority over other competing methods. First, a geometrically complex domain is represented as a collection of geometrically simple subdomains, called finite elements. Second, over each finite element the approximating functions are usually chosen polynomial. The approximating functions are derived using concepts from interpolation theory, and are therefore called interpolation functions.

Consider a region Ω which is approximated by Ω_h with m four-node tetrahedral elements (Fig. 3.5) K_i,

$$\Omega_h = \bigcup_{i=1}^{m} K_i.$$

The mesh size h is defined by

$$h = \max_{i=1,\ldots,m} \text{diam}(K_i)$$

where diam(K), the diameter of a tetrahedral element K, is the length of its longest side.

Then the finite dimensional space V_h is defined as the space of all functions which are piecewise linear and continuous on the polygonal region Ω_h, and which vanish on the boundary Γ. More specifically,

$$V_h = \left\{\phi \mid \phi_{|\Omega_h} \text{ continuous}, \ \phi_{|\Gamma_h} = 0, \phi_{|K_j} \text{ linear } \forall j\right\}.$$

Here, $\phi_{|X}$ represents the restriction of the function ϕ to the subset X. If x_j, $j = 1,\ldots,n$ are the nodes of the triangulation, then a function ϕ_j in V_h can be associated with each node x_j, such that the family of functions ϕ_j's satisfies the following conditions:

$$\phi_j(x_i) = \delta_{ij} = \begin{cases} 1 & \text{if } x_i = x_j \\ 0 & \text{if } x_i \neq x_j \end{cases}.$$

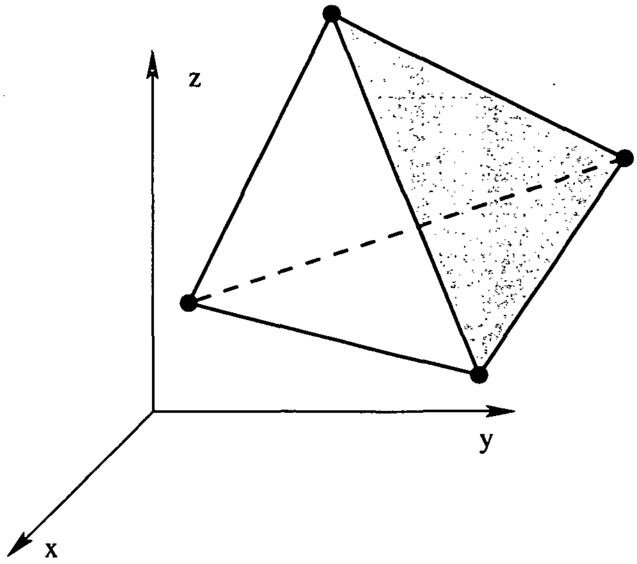


Figure 3.5: Four-point tetrahedral element.

These conditions define ϕ_i, $i = 1, \ldots, n$ uniquely. In addition, the ϕ_i's form a basis of the space V_h.

Each function of V_h can be expressed as

$$\phi(x) = \sum_{i=1}^{n} \xi_i \, \phi_i(x).$$

The finite element approximation consists of writing the Galerkin condition for functions in V_h:

$$\text{Find } u \in V_h \text{ such that } a(u,v) = (f,v), \quad \forall v \in V_h, \tag{3.6}$$

with the bilinear functional

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \; dx,$$

and (f,v) denoting the L_2-inner product of u and v in Ω,i. e.,

$$(f,v) = \int_\Omega f \, v \; dx.$$

Since u is in V_h, there are n degrees of freedom. By the linearity of a with respect to v, it is only necessary to impose the condition $a(u, \phi_i) = (f, \phi_i)$ for $i = 1, \ldots, n$. This results in n constraints.

Writing the desired solution u in the basis $\{\phi_i\}$ as

$$u = \sum_{i=1}^{n} \xi_i \, \phi_i(x)$$

and substituting into (3.6) gives the linear problem

$$\sum_{j=1}^{n} \alpha_{ij} \, \xi_j = \beta_i$$

where

$$\alpha_{ij} = a(\phi_i, \phi_j), \quad \beta_i = (f, \phi_i).$$

The above equations form a linear system of equations

$$Ax = b$$

in which the coefficients of A are the α_{ij}'s and those of b are the β_i's. It can be shown that for the problem considered A is again a sparse, symmetric and positive definite matrix.

The are a few advantages of the finite element method for the application of functional electrical stimulation (FES) beside the ones mentioned in the beginning.

Coarseness Control The most important tissue type in FES are muscles and depending on the type of functional activation often a specific group of muscles are of special interest. The finite element method allows to define a non uniform mesh-size and when using a fine grained mesh in the zones of interest this will lead to more accurate results.

Smaller Systems When using a fine grained mesh at specific areas it is also possible to use larger elements and therefore fewer data to represent tissue like bone or fat or the areas near the knee and the hip. This would lead to an overall smaller system and faster calculation time.

User Interface There exist numerous tools for modeling and simulating PDEs which also often provide a graphical user interface for ease of use (e. g., FEMLAB [19]). These tools can be used to build up the mesh structure with a convenient user interface to specify the zones of interest or other areas with a more coarse grid. After solving the delineated problem these programs also provide sophisticated tools to analyze the solution.

3.2 Matrix Properties

Since the original data is available as uniform mesh and because of the simpler implementation in the current implementation of the FES-Tool at the Department of Biomedical Engineering and Physics the Finite Difference Method is used. A complete physical derivation of the problem can be found at Breyer [11] or Grotz [25]. After building up the system and removing rows / columns containing only zeros (= the air in the cuboid) matrices have a sparsity structure as depicted in Fig. 3.6 (nonzero entries are shown as dots). Since the small and large matrices share common properties some examples in the following are only tested with the small datasets when runtime exceeds limits for the bigger problems.

Data is available as CT images and only a fraction of the information is used to build the conductance matrix because otherwise the matrices would be too big. The scaling factor therefore determines the size of the system which is usually given in the notation *height*×*width*×*depth*, i. e., the images in Fig. 3.6 represent a resolution of 3 slides with 105×89 conductance values each respectively 51 slides with a resolution of 314×266 voxel.

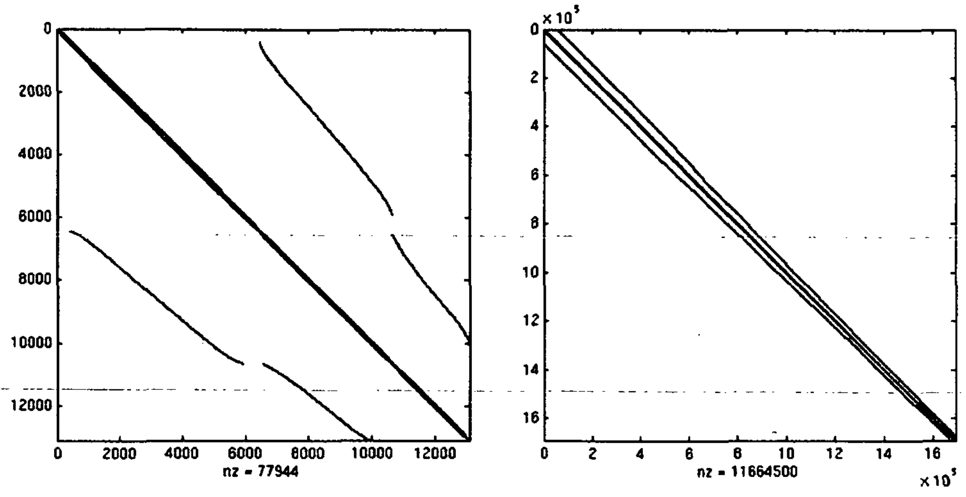


Figure 3.6: Sparsity structure of matrices with the size 105 x 89 x 3 (left) and 314 x 266 x 51 (right).

Some immediate properties of this matrices are:

Dimension The size of the investigated matrices is between $1.3 \cdot 10^4 - 1.6 \cdot 10^6$. An rough estimate of the size is about 40% of *height*×*width*×*depth* which means that the rest is usually air.

Sparsity From the definition of the matrix it follows that there are at most seven nonzero entries in each row, i. e., the number of nonzeros (nnz) is about 7×dim. In the examples above we have nnz=77 944 respectively nnz=11 664 500 or a sparsity ($\frac{\text{nnz}}{\text{dim}^2}$) of $4.5 \cdot 10^{-4}$ / $4.1 \cdot 10^{-6}$.

Bandwidth An upper bound for the bandwidth defined as

$$\max\{|i-j| \mid a_{ij} \neq 0\} \qquad \text{with } A = \{a_{ij}\}$$

follows again from the matrix definition. The conductivity to the left and right voxel is immediately next to the main diagonal. Values for the voxels above and below are *with* columns afar from the main diagonal and voxels in front and behind have a distance of *height* × *width*. The above border is therefore 1+*width*+*height*×*width*. Because of the removal of air 40% of this value is an approximation of the real bandwidth: 6 056 respectively 58 217 for the systems above.

Matrices of the type considered here are saved in a special sparse format (usually the *Compressed Sparse Row* (CSR) format). MATLAB for instance takes about 1MB for a small system and 146MB for a large system to save the system in CSR format—for comparison, the small system in conventional storage (as dense matrix) would need 1.4GB.

Another important characteristic is that bandwidth reduction algorithms can be successfully applied. The reverse Cuthill McKee algorithm (Cuthill and McKee [16]) for example reduces the bandwidth significantly which leads to numbers of 173 respectively 8 027 for the above systems which is especially important for some solution strategies as will be shown in the next chapter.

Further investigation reveals the following numerical properties of the systems.

Definiteness The matrices are symmetric and negative definite. By construction the sum of each row / column is 0 because the diagonal element is the negative sum of all elements in the row column. This follows from the first Kirchhoff's current law which states that the sum of all currents flowing into a node is zero.

For the purpose of solving the systems it is convenient to consider positive definite systems and therefore in the following $A := -A$ is assumed. An $n \times n$ symmetric matrix A is called positive definite if

$$x^T A x > 0$$

for all nonzero $x \in \mathbb{R}^n$. This is equivalent to the requirement that all eigenvalues are positive, or to the requirement that determinants associated with all upper-left sub-matrices are positive. Another *sufficient* requirement is that the matrix $A = (a_{ij})$ is diagonal dominant

$$\sum_{i \neq j} |a_{ij}| < |a_{ii}|,$$

or irreducibly diagonaldominant, i. e., A is irreducible and

$$\sum_{i \neq j} |a_{ij}| \leq |a_{ii}|$$

with at least one index i fulfilling the strict inequality. The proof for positive definiteness of the matrices arising in the context of functional electrical stimulation can be found for instance in Auzinger [6].

Condition of the Problem The condition number of a matrix measures the worst case of the sensitivity of the solution of a system of linear equations to errors in the data. It also provides an indication of the accuracy of the results from matrix inversion and the linear equation solution.

The condition number of a regular quadratic matrix A with respect to a norm p is defined as

$$\kappa_p(A) = \|A\|_p \, \|A^{-1}\|_p. \tag{3.7}$$

Values of $\kappa(A)$ near 1 indicate a well conditioned matrix and poorly conditioned matrices have large numbers of $\kappa(A)$.

But it is usually expensive to calculate the κ and it takes several hours to calculate the condition number even for a small system if the inverse is explicitly calculated. Therefore tools like MATLAB provide special commands for approximating the condition number (e. g., `condest` computes a lower bound for the 1-norm condition number).

For a small system $(105 \times 89 \times 3)$ $\kappa = 6.9 \cdot 10^5$ and with increasing size the condition gets worse which indicates the need for good preconditioners as will be seen in the next chapters.

Taking all this into account one finds that today there exists a wide variety of solution methods for solving such systems and solvers for linear systems are indeed one of the best studied fields in numerical linear algebra. But choosing a solution method for a given problem does not depend only on the mathematical properties of the problem. The field where the problem solving environment has to be applied also has to be taken into account. For the FES-Tool this includes currently the following restrictions.

Implementation in MATLAB The FES-Tool is implemented in MATLAB (Higham and Higham [28]) using the rich functionality of functions for general problems. Therefore, also the solver has to be implemented in MATLAB or be invoked from there and should return the result to MATLAB for postprocessing of the result. Because of the size of the problems we have to use a sparse storage format as stated above and using MATLAB this naturally suggests the *Compressed Sparse Row* (CSR) format.

Single processor machine running Windows Currently most of the research in high performance computing is done in distributed computing and especially Grid computing (Foster and Kesselman [21]) and much progress has happened in the field of solving large system on distributed resources. Nevertheless, the aim of solving is in this case on the use of just a single machine running the Windows operating systems with 3GB of main memory.

Solving $Ax = b$ In the FES-Tool solving the problem of calculating a current distribution within the thigh is just one part of the much bigger project. So, for easy interoperability between the various parts a simple interface was designed which specifies that only the matrix A and the right-hand-side vector b are provided and as result the solution vector x should be returned.

Another restriction, although not explicitly specified above, is the use of available and proven implementations of solvers. Since these sophisticated tools take years to develop and tune and also require a great deal of expert knowledge and experience in this field, it is often better to use these tools instead of implementing them from scratch.

From the mathematical point of view the following properties of the system have to be taken into account.

- Large, sparse systems recommend an iterative solution strategy and
- the matrices are symmetric and positive definite, thus some sort of conjugate gradient method is the main choice.

Given all these determining factors it turns out that the availability of free software on the Windows platform is quite limited and development takes manly place on Unix. Fortunately there is a tool available on Windows which allows the compilation and execution of software from UNIX (especially Linux) called Cygwin (Vinschen et al. [55]). Together with the MEX interface of Matlab (MATLAB: Application Program Interface Guide [39]), which allows the execution of arbitrary C and Fortran Code, it is possible to access this software from Windows, too. If the restriction of running the solver on the machine at the Department of Biomedical Engineering and Physics is additionally softened it will be possible to use tools like NetSolve (Arnold et al. [3]) or HARNESS (Fagg et al. [18]) to remotely execute software on any machine connected to the Internet.

The following two chapters will now describe the mathematics of the solvers used, i. e., the Krylov Subspace Methods in Chapter 4 and multigrid methods for preconditioning in Chapter 5.

Chapter 4

Solution Methods

This chapter describes the methods for solving the systems already mentioned in Chapter 3. We will start with a short introduction to applicable direct solvers and will turn afterwards to iterative methods following mainly the presentation in *Iterative Methods for Sparse Linear Systems* by Saad [47]. Starting with basic iterative methods like Jacobi and Gauss-Seidel we proceed into projection methods and finish with Krylov subspace methods (for further details see especially *Iterative Krylov Methods for Large Linear Systems* by van der Vorst [57]).

As example problem for this presentation, the classical model for a discrete elliptic boundary value problem is used. The discrete Poisson equation with Dirichlet boundary conditions

$$\begin{aligned} -\Delta u(\boldsymbol{x}) &= f(\boldsymbol{x}) && \text{for } \boldsymbol{x} = (x_1, x_2, x_3), \boldsymbol{x} \in \Omega = (0,1)^3 \subset \mathbb{R}^3 \\ u(\boldsymbol{x}) &= \varphi(\boldsymbol{x}) && \text{on } \Gamma = \partial\Omega. \end{aligned} \tag{4.1}$$

For discretization of the differential equation a uniform mesh with mesh size $h = 1/n$ is used. The mesh is the set of inner points

$$\Omega_h = \{(x_1, x_2, x_3) = (ih, jh, kh) \mid 1 \leq i, j, k \leq n - 1\}.$$

For an approximation of the differential equation the seven-point stencil (Figure 3.2) is used leading to a linear system $A\,x = b$ (cf. 3.5). When iteratively solving this system the i-th component of the solution vector after k iterations is denoted by $x_i^{(k)}$.

For the sake of completeness the following section describes available direct solvers for large, sparse, linear systems. But as shown in the results this type of solver can't compete against the much more efficient preconditioned conjugate gradient methods for the given problem.

4.1 Sparse Direct Solvers

A direct method for solving a linear system is Gaussian elimination, i. e., factoring a matrix into the product of a unit lower triangular matrix and an upper triangular matrix. The problem for sparse systems is that during elimination many nonzero elements are generated. Sparse direct solvers therefore seek to apply a good reordering to avoid this fill-in—a method which can improve a direct solver significantly, but does very little to the efficiency of most iterative algorithms. Algorithm 1 gives a general picture how a sparse direct solver works.

Algorithm 1 A simple sparse Gaussian elimination.

---

1. Compute a triangular factorization $P_r\,A\,P_c = L\,U$, with P_r and P_c permutations matrices, where P_r reorders the rows of A and post-multiplying by P_c reorders the columns. P_r and P_c are chosen to retain sparsity (avoid fill-in) and numerical stability, e. g., the multiple minimum degree ordering MMD (Liu [36]) or the column approximate minimum degree ordering COLAMD (Davis et al. [17]) could be used.
2. Solve $A\,x = b$ by evaluating $x = A^{-1}\,b = \left(P_c^{-1} L U P_r^{-1}\right)^{-1} b = P_c \left(U^{-1} \left(L^{-1}(P_r\,b)\right)\right)$. This is done efficiently by multiplying from right to left in the last expression, where multiplying with P means a permutation and multiplying by an inverse means solving the system.

The following two solvers have been chosen as examples since they are quite prominent and established in the scientific community.

SuperLU The SuperLU package is a collection of three related ANSI C subroutine libraries for solving sparse linear systems of equations. It contains a set of subroutines to solve sparse linear systems $AX = B$. Here A is a square, nonsingular, $n \times n$ sparse matrix, and X and B are dense $n \times nrhs$ matrices, where $nrhs$ is the number of right-hand sides and solution vectors. The matrix A need not to be symmetric or definite; indeed, SuperLU is particularly appropriate for matrices with very unsymmetric structure.

All three libraries use variations of Gaussian elimination optimized to take advantage both of sparsity and the computer architecture, in particular memory hierarchies (caches) and parallelism. The three libraries within SuperLU are *Sequential SuperLU*, *Multithreaded SuperLU* and *Distributed SuperLU*. For the problems in the thesis only Sequential SuperLU was tested for performance against other algorithms—see Chapter 7 for results.

MA28 / MA48 The Harwell Sparse Matrix Library contains routines for handling sparse linear and nonlinear problems selected from the Harwell Subroutine Library and is part of the Numerical Algorithms Group (NAG) package. There is now a second release of the libraries available with significant upgrades and extensions. The MA48 library solves a sparse unsymmetric system of linear equations using Gaussian elimination. There are facilities for block triangularization, iterative refinement and error estimation. For tests in this work only MA28 was available, but as numerical experiments in the literature indicate, e. g., Barton et al. [8], the newer MA48 is on average about 20 – 30% faster than MA28.

4.2 Basic Iterative Methods

The first iterative methods used for solving large linear systems were based on relaxation of the coordinates. Beginning with a given approximate solution, these methods modify the components of the approximation, one or a few at a time in a certain order, until convergence is reached. Each of these modifications, called relaxation steps, is aimed at annihilating one or a few components of the residual vector. The methods covered in this section involve passing from one iterate to the next by modifying a component in order to improve an iterate through eliminating some component(s) of the residual vector $b - Ax$.

The Jacobi iteration determines the i-th component of the next approximation so as to annihilate the i-th component of the residual vector—see Algorithm 2.

Algorithm 2 General Jacobi Iteration.

```
1. Until convergence do $k = k + 1$:
2.     $x_i^{(k+1)} = x_i^k - \frac{1}{a_{ii}} \left( b_i - \sum_{j=1, j \neq i}^{n} a_{ij} \, x_j^{(k)} \right) \qquad i = 1, \ldots, n$
```

Similarly, the Gauss-Seidel iteration corrects the i-th component of the current approximate solution, in the order $i = 1, \ldots, n$, again to annihilate the i-th component of the residual. However, this time the approximation is updated immediately after the new component is determined. The newly computed components $x_i^{(k)}$ can be changed within a working vector which is redefined at each relaxation step—see Algorithm 3.

Algorithm 3 General Gauss-Seidel Iteration.

```
1. Until convergence do $k = k + 1$:
2.     $x_i^{(k+1)} = x_i^k - \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} \, x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} \, x_j^{(k)} \right) \qquad i = 1, \ldots, n$
```

Successive Over Relaxation (SOR) finally adds a weight factor ω to the expression in parenthesis of the Gauss-Seidel iteration (Algorithm 4). A good choice for this relaxation factor can speed up the convergence significantly—see Hackbusch [27] for furhter details. In Table 4.1 results of the discussed algorithms are compared in terms of convergence rate for the model problem.

Algorithm 4 General SOR Iteration.

```
1. Until convergence do $k = k + 1$:
2.     $x_i^{(k+1)} = x_i^k - \frac{1}{a_{ii}} \omega \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \qquad i = 1, \dots, n$
```

As example for comparing these basic iterative methods the problem

$$f_{ijk} = -6, \quad \varphi(\boldsymbol{x}) = x_1^2 + x_2^2 + x_3^2 \tag{4.2}$$

is used. For these data the solution is $u_h(\boldsymbol{x}) = x_1^2 + x_2^2 + x_3^2$. Table 4.1 shows for $h = 1/32$ ($\leadsto$ dim $A = 29\,791$) the value

$$p_m = u^m_{16,16,16} \tag{4.3}$$

in the middle $(16h, 16h, 16h) = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ which should converge to 0.75, and the error

$$\varepsilon_m = \max\left\{ |u^m_{ijk} - (i^2 + j^2 + k^2)h^2| \mid 1 \le i, j, k \le n - 1 \right\} \tag{4.4}$$

after m iterations.

	Jacobi		Gauss-Seidel		SOR ($\omega = 1.821$)	
Itr	p_m	ε_m	p_m	ε_m	p_m	ε_m
1	−0.001	$2.637 \cdot 10^{0}$	−0.002	$2.639 \cdot 10^{0}$	−0.016	$3.622 \cdot 10^{0}$
50	−0.048	$1.428 \cdot 10^{0}$	−0.070	$1.142 \cdot 10^{0}$	0.744	$2.296 \cdot 10^{-2}$
100	−0.062	$1.079 \cdot 10^{0}$	0.083	$7.231 \cdot 10^{-1}$	0.749	$4.691 \cdot 10^{-6}$
500	0.586	$1.640 \cdot 10^{-1}$	0.734	$1.565 \cdot 10^{-2}$	0.750	$2.220 \cdot 10^{-15}$
1000	0.735	$1.471 \cdot 10^{-2}$	0.750	$1.254 \cdot 10^{-4}$	0.750	$2.220 \cdot 10^{-15}$
2000	0.750	$1.178 \cdot 10^{-4}$	0.750	$8.046 \cdot 10^{-9}$	0.750	$2.220 \cdot 10^{-15}$

Table 4.1: Results of the basic iterative methods for the model problem.

The convergence rate of Jacobi and Gauss-Seidel is very poor and only the SOR method converges within the first hundred iterations to a reasonably small error. The Jacobi iteration has some appeal on parallel computers because the approximation is updated only in the outer *do* loop. However, these techniques are rarely used separately for real-life problems but, combined with more efficient methods described later, they can be quite useful.

4.3 Projection Methods

Most existing practical iterative techniques for solving large linear systems of equations utilize a projection process in one way or another. A projection process represents a canonical way for extracting an approximation to the solution of a linear system from a subspace. If $\mathcal{K}$ is this subspace of candidate approximants and if m is its dimension, then, in general m constraints must be imposed to be able to extract such an approximation. A typical way of describing these constraints is to impose m (independent) orthogonality conditions. Specifically, the residual vector $b - Ax$ is often constrained to be orthogonal to m linearly independent vectors. This defines another subspace $\mathcal{L}$ of dimension m which is called the subspace of constraints. This simple framework is common to many different mathematical methods and is known as the Petrov-Galerkin conditions. When $\mathcal{L} = \mathcal{K}$, the Petrov-Galerkin conditions are called the Galerkin conditions.

For example in the simplest case an elementary Gauss-Seidel step could be seen as a projection step with $\mathcal{L} = \mathcal{K} = \text{span}\{e_i\}$. Another example is the steepest descent algorithm for symmetric positive definite systems which is also an one-dimensional projection process in each iteration step. One-dimensional projection processes are defined when

$$\mathcal{K} = \text{span}\{v\} \quad \text{and} \quad \mathcal{L} = \text{span}\{w\},$$

where v and w are two vectors. In this case the new approximation takes the form $x^{(k+1)} = x^{(k)} + \alpha v$ and the Petrov-Galerkin condition $r - A(\tilde{x} - x^{(0)}) \perp w$ yields

$$\alpha = \frac{(r, w)}{(Av, w)}.$$

In the steepest descent algorithm we have $v = r$ and $w = r$ in each step. This leads to an iteration described in Algorithm 5.

Algorithm 5 Steepest Descent Iteration.

```
1. Until convergence do $k = k + 1$:
2.     $r = b - Ax$
3.     $\alpha = (r, r)/(Ar, r)$
4.     $x^{(k+1)} = x^{(k)} + \alpha r$
```

This algorithm can be enhanced to the non-symmetric case (minimal residual iteration) and the general non-singular case (residual norm steepest descent). These techniques are the basis for the Krylov subspace methods which are considered currently to be among the most important iterative techniques and are described in the next section.

4.4 Krylov Subspace Methods

A Krylov subspace method is a projection method where the subspace $\mathcal{K}$ is a Krylov subspace

$$\mathcal{K}_m(A, r_0) = \text{span}\{r_0, Ar_0, A^2 r_0, \dots, A^{m-1} r_0\},$$

with $r_0 = b - Ax_0$. The different versions of Krylov subspace methods arise from different choices of the subspace $\mathcal{L}_m$ and from the ways in which the system is preconditioned—see Chapter 5.

The motivation for using an m-dimensional Krylov subspace can be derived from the well-known Richardson iteration

$$x_m = b + (I - A)x_{m-1} = x_{m-1} + r_{m-1}.$$

By repeating this iteration it is observed that

$$\begin{aligned} x_m &= r_0 + r_1 + r_2 + \cdots + r_{m-1} \\ &= \sum_{j=0}^{m-1} (I - A)^j \, r_0 \\ &\in \text{span}\{r_0, Ar_0, A^2 r_0, \dots, A^{m-1} r_0\} \\ &\equiv \mathcal{K}_m(A, r_0). \end{aligned}$$

From the point of view of approximation theory the approximations obtained from a Krylov subspace method are of the form

$$A^{-1} b \approx x_m = x_0 + q_{m-1}(A) r_0,$$

in which q_{m-1} is a certain polynomial of degree $m-1$. In the simplest case where $x_0 = 0$ we have $A^{-1}b \approx q_{m-1}(A)b$ and in other words, $A^{-1}b$ is approximated by $q_{m-1}(A)b$. From now on it is always assumed that $x_0 = 0$ to simplify future formulas. This does not mean a loss of generality, because the situation $x_0 \neq 0$ can be transformed with a simple shift to the system

$$Ay = b - Ax_0 = \bar{b}$$

for which obviously $y_0 = 0$.

Apparently, the Richardson iteration, as it proceeds, delivers elements of Krylov subspaces of increasing dimension and therefore the Krylov subspace has to be explored. The projection methods discussed here focus on the case where $\mathcal{L}_m = \mathcal{K}_m$.

Two important algorithms based on the Krylov subspace methods, for identifying suitable $x \in \mathcal{K}_m(A, r_0)$ are

1. General Minimum Residual Method (GMRES)—with the minimum norm residual approach: Identify the x_m for which the Euclidean norm $\|b - Ax_m\|_2$ is minimal over $\mathcal{K}_m(A, r_0)$.

2. Conjugate Gradient Method (CG)—with the Ritz-Galerkin approach: Construct the x_m for which the residual is orthogonal to the current subspace: $b - Ax_m \perp \mathcal{K}_m(A, r_0)$.

A suitable basis for the Krylov subspace is needed in order to identify the approximations of the above approaches. The obvious basis $r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0$ for $\mathcal{K}_m(A, r_0)$, is not very attractive from a numerical point of view, since the vectors $A^j r_0$ point more and more in the direction of the dominant eigenvector for increasing j, and hence the basis vectors become dependent in finite precision arithmetic.

Arnoldi [4] proposed to compute the orthogonal basis as follows. Start with $v_1 \equiv r_0/\|r_0\|_2$. Then compute Av_1, make it orthogonal to v_1 and normalize the result, which gives v_2. The general procedure is as follows. Assuming an already orthonormal basis $v_1, \ldots, v_j$ for $\mathcal{K}_m(A, r_0)$, this basis is expanded by computing $t = Av_j$ and by orthonormalizing this vector t with respect to $v_1, \ldots, v_j$. In principle the orthonormalization process can be carried out in different ways, but the most commonly used approach is the modified Gram-Schmidt procedure (Golub and Van Loan [24])—see Algorithm 6.

Algorithm 6 Arnoldi-Modified Gram-Schmidt.

```
1. $v_1 = r_0/\|r_0\|_2$
2. For $j = 1, 2, \ldots, m-1$ do:
3.     $t = Av_j$
4.     For $i = 1, 2, \ldots, j$ do:
5.         $h_{ij} = v_i^T t$
6.         $t = t - h_{ij} v_i$
7.     $h_{j+1,j} = \|t\|_2$
8.     $v_{j+1} = t/h_{j+1,j}$
```

4.4.1 GMRES algorithm

The creation of an orthogonal basis for the Krylov subspace, with basis vectors $v_1, \ldots, v_{m+1}$, leads to

$$AV_m = V_{m+1} H_{m+1,m},$$

where V_m is the matrix with columns v_1 to v_m and the $m+1$ by m matrix $H_{m+1,m}$ is upper Hessenberg with its elements h_{ij} defined by the Arnoldi algorithm. An $x_m \in \mathcal{K}_m(A, r_0)$ is now sought, that is $x_m = V_m y$, for which $\|b - Ax_m\|_2$ is minimal. This norm can be rewritten, with $\rho \equiv \|r_0\|_2$, as

$$\|b - Ax_m\|_2 = \|r_0 - AV_m y\|_2 = \|\rho V_{m+1} e_1 - V_{m+1} H_{m+1,m} y\|_2.$$

Exploiting the fact that V_{m+1} is an orthonormal transformation with respect to the Krylov subspace $\mathcal{K}_{m+1}(A, r_0)$ leads to

$$\|b - Ax_m\|_2 = \|\rho e_1 - H_{m+1,m} y\|_2,$$

and this final form can be minimized by solving the minimum norm least square problem for the $m + 1$ by m matrix $H_{m+1,m}$ and right-hand side $\|r_0\|_2 \, e_1$. The least squares problem is solved by constructing a QR factorization of $H_{m+1,m}$, and because of the upper Hessenberg structure that can conveniently be handled with Givens transformations (Golub and Van Loan [24]).

The Givens rotations annihilate the subdiagonal elements in the upper Hessenberg matrix $H_{m+1,m}$. The resulting upper triangular matrix is denoted by R_{mm}:

$$H_{m+1,m} = Q_{m+1,m} R_{mm},$$

where $Q_{m+1,m}$ denotes the product of successive Givens eliminations of the elements $h_{j+1,j}$, for $j = 1, \ldots, m$. After the Givens transformations the least squares solution y minimizes

$$\|H_{m+1,m} y - \|r_0\|_2 e_1\|_2 = \|Q_{m+1,m} R_{mm} y - \|r_0\|_2 e_1\|_2 = \|R_{mm} y - Q^T_{m+1,m} \|r_0\|_2 e_1\|_2.$$

The resulting least squares problem leads to the minimum norm solution

$$y = R^{-1}_{mm} \, Q^T_{m+1,m} \, \|r_0\|_2 \, e_1.$$

The required approximation x_i is now computed as $x_i = V_i y$.

In order to avoid excessive storage requirements and computational costs for the orthogonalization, GMRES is usually restarted after n iteration steps. This algorithm is referred to GMRES(n). The not restarted version is often called 'full' GMRES. There is no simple rule to determine a suitable value for n and the speed of convergence may vary drastically for nearby values of n. It may be the case that GMRES($n + 1$) is much more expensive than GMRES(n), even in terms of number of iterations. Algorithm 7 shows the modified Gram-Schmidt version of GMRES(n) for the solution of the linear system $Ax = b$.

Algorithm 7 General Minimum Residual (GMRES) algorithm with restarting after n steps.

```
$r = b - Ax_0$, for a given initial guess $x_0$
$x = x_0$
For $j = 1, 2, \dots$ do:
    $\beta = \|r\|_2$, $v_1 = r/\beta$, $\widehat{\beta} = \beta e_1$
    For $i = 1, 2, \dots, n$ do:
        $w = Av_i$
        For $k = 1, 2, \dots, i$ do:
            $h_{ki} = v_k^T w$, $w = w - h_{ki} v_k$
        $h_{i+1,i} = \|w\|_2$, $v_{i+1} = w/h_{i+1,i}$, $r_{1i} = h_{1i}$
        For $k = 2, 3, \dots, i$ do:
            $\gamma = c_{k-1} r_{k-1,i} + s_{k-1} h_{ki}$, $r_{ki} = -s_{k-1} r_{k-1,i} + c_{k-1} h_{ki}$, $r_{k-1,i} = \gamma$
        $\delta = \sqrt{r_{ii}^2 + h_{i+1,i}^2}$, $c_i = r_{ii}/\delta$, $s_i = h_{i+1,i}/\delta$, $r_{ii} = c_i r_{ii} + s_i h_{i+1,i}$
        $\widehat{b}_{i+1} = -s_i \widehat{b}_i$, $\widehat{b}_i = c_i \widehat{b}_i$, $\rho = |\widehat{b}_{i+1}|$
        If $\rho$ is small enough then goto SOL
    $y_n = \widehat{b}_n / r_{nn}$
SOL: For $k = i-1, i-2, \dots, 1$
        $y_k = (\widehat{b}_k - \sum_{l=k+1}^{i} r_{kl} y_l)/r_{kk}$
    $x = x + \sum_{l=1}^{i} y_l v_l$
    If $\rho$ small enough then quit
    $r = b - Ax$
```

4.4.2 Conjugate Gradient algorithm

As noted above the Conjugate Gradient method uses the Ritz-Galerkin conditions which imply that $r_m \perp \mathcal{K}_m(A, r_0)$, and this is equivalent to

$$V_m^T(b - Ax_m) = 0.$$

Since $b = r_0 = \|r_0\|_2 v_1$, it follows that $V_m^T b = \|r_0\|_2 e_1$, and with $x_m = V_m y$ it leads to

$$V_m^T A V_m y = \|r_0\|_2 e_1.$$

This system can be interpreted as the system $Ax = b$ projected onto the subspace $\mathcal{K}_m(A, r_0)$. Obviously the $m \times m$ matrix $V_m^T A V_m$ has to be constructed, but this is readily available from the orthogonalization process (see Algorithm 6)

$$V_m^T A V_m = H_{mm},$$

so that the x_m for which $r_m \perp \mathcal{K}_m(A, r_0)$ can be computed by first solving $H_{mm} y = \|r_0\|_2 \, e_1$, and then forming $x_m = V_m y$. This algorithm is also known as Full Orthogonalization Method (FOM) or General Conjugate Gradient Method (GENCG)—see Saad and Schultz [48].

When A is symmetric, then H_{mm} reduces to a tridiagonal matrix T_{mm}, and the resulting method is known as the Lanczos method (Lanczos [33]). When A is in addition positive definite then formally the Conjugate Gradient method is obtained. In commonly used implementations of this method, an LU factorization for T_{mm} is implicitly formed without generating T_{mm} itself. (The positive definiteness is necessary to guarantee the existence of the LU factorization.)

In the Galerkin approach, the new residual $b - Ax_{m+1}$ is orthogonal to the subspace spanned by $v_1, \ldots, v_m$, so that r_{m+1} is in the direction of v_{m+1}. Therefore, the scaling factor $h_{m+1,m}$ can also be selected that v_{m+1} coincides with r_{m+1}. This would be convenient, since the residual gives useful information on the solution, and there do not have to be work done with two sequences of auxiliary vectors.

If R_i denotes the matrix with columns r_j it leads to the relation

$$AR_m = R_{m+1} T_{m+1,m}, \tag{4.5}$$

with T the tridiagonal matrix from above. Additionally, since the solution for x_m in $\mathcal{K}_m(A, r_0)$ is sought, that vector can be written as a combination of the basis vectors of the Krylov subspace, and hence

$$x_m = R_m y.$$

Furthermore, the Ritz-Galerkin condition says that the residual for x_m is orthogonal with respect to $r_0, \ldots, r_{m-1}$:

$$R_m^T (Ax_m - b) = 0,$$

and hence

$$R_m^T A R_m y - R_m^T b = 0.$$

Using (4.5) leads to

$$R_m^T R_m T_m y = \|r_0\|_2^2 \, e_1,$$

where $R_m^T R_m$ is a diagonal matrix with diagonal elements $\|r_0\|_2^2$ up to $\|r_{m-1}\|_2^2$. The desired solution can be found by solving y from

$$T_{mm} y = e_1 \quad \Rightarrow \quad y \quad \Rightarrow \quad x_m = R_m y.$$

So far only the fact that A is symmetric was used and it was assumed that the matrix T_m is not singular. The Conjugate Gradient method is now a clever variant on the above approach that saves storage and computational effort. In the way described above all columns of R_m throughout the process would have to be saved in order to recover the current iteration vectors. This can be done in a more memory friendly way. If the matrix A is in addition assumed to be positive definite and with the relation

$$R_m^T A R_m = R_m^T R_m T_m,$$

T_m can be transformed by a rowscaling matrix $R_m^T R_m$ into a positive definite symmetric tridiagonal matrix. This implies that T_m can be LU decomposed without any pivoting:

$$T_m = L_m U_m,$$

with L_m lower bidiagonal, and U_m upper diagonal with unit diagonal.

Defining $P_m \equiv R_m U_m^{-}1$ leads to the auxiliary vectors p_j which form an A-conjugate set, i. e., $(Ap_i, p_j) = 0$, for $i \neq j$. This follows from the fact that $P_m^T A P_m$ is a diagonal matrix:

$$\begin{aligned} P_m^T A P_m &= U_m^{-T} V_m^T A V_m U_m^{-1} \\ &= U_m^{-T} T_m U_m^{-1} \\ &= U_m^{-T} L_m. \end{aligned}$$

Observe that $U_m^{-T} L_m$ is lower triangular and also symmetric since equal to the symmetric matrix $P_m^T A P_m$ it must be therefore a diagonal matrix.

The final algorithm can now be derived by imposing the orthogonality of the residual vectors to each other and the conjugacy conditions of the p_i's. The vector x_{j+1} can be expressed as

$$x_{j+1} = x_j + \alpha_j p_j.$$

Therefore, the residual vectors must satisfy the recurrence

$$r_{j+1} = r_j - \alpha_j A p_j. \tag{4.6}$$

If the r_j's are to be orthogonal, then it is necessary that $(r_j - \alpha_j A p_j, r_j) = 0$ and as a result

$$\alpha_j = \frac{(r_j, r_j)}{(Ap_j, r_j)}. \tag{4.7}$$

Also it is known that the next search direction p_{j+1} is a linear combination of r_{j+1} and p_j, and after rescaling the p vectors appropriately, it follows that

$$p_{j+1} = r_{j+1} + \beta_j p_j. \tag{4.8}$$

Thus a first consequence of the above relation is that

$$(Ap_j, r_j) = (Ap_j, p_j - \beta_{j-1} p_{j-1}) = (Ap_j, p_j)$$

because Ap_j is orthogonal to p_{j-1}. Then, (4.7) becomes $\alpha_j = (r_j, r_j)/(Ap_j, p_j)$. In addition writing that p_{j+1} as defined by (4.8) is orthogonal to Ap_j yields

$$\beta_j = -\frac{(r_{j+1}, Ap_j)}{(p_j, Ap_j)}.$$

Note that from (4.6)

$$Ap_j = -\frac{1}{\alpha_j}(r_{j+1} - r_j)$$

and therefore,

$$\beta_j = \frac{1}{\alpha_j} \frac{(r_{j+1}, (r_{j+1} - r_m j))}{(Ap_j, p_j)} = \frac{(r_{j+1}, r_{j+1})}{(r_j, r_j)}.$$

Putting theses relations together leads to Algorithm 8.

Algorithm 8 Conjugate Gradient algorithm.

```
1. $r_0 = b - Ax_0$, $p_0 = r_0$
2. Until convergence do $k = k + 1$:
3.     $\alpha_k = (r_k, r_k)/(Ap_k, p_k)$
4.     $x^{(k+1)} = x^{(k)} + \alpha_k p_k$
5.     $r_{k+1} = r_k - \alpha_k A p_k$
6.     $\beta_k = (r_{k+1}, r_{k+1})/(r_k, r_k)$
7.     $p_{k+1} = r_{k+1} + \beta_k p_k$
```

4.4.3 Overview

The choice of a method for a given problem is a delicate problem. If the matrix A is symmetric positive definite, then the choice is usually easy: Conjugate Gradients. For other types of matrices the situation is very diffuse. GMRES, proposed 1986 by Saad and Schultz [48], is the most robust method, but in terms of work per iteration step it is also relatively expensive. Bi-CG, which was suggested by Fletcher [20] in 1976, is a relatively inexpensive alternative, but it has problems with respect to convergence: the so-called breakdown situations.

The development of hybrid methods started with CGS, published in 1989 by Sonneveld [51], and was followed by BI-CGSTAB, by van der Vorst [56] in 1992. The hybrid variants of GMRES: Flexible GMRES and GMRESR, in which GMRES is combined with some other iteration scheme, were proposed in the mid-1990s.

Table 4.2 shows the results of solving the model problem as described in Section 4.2 for the presented projection methods. Steepest Descent performs as expected in the range of the basic iterative Methods comparable to Jacobi. The GMRES(10) method seems to converge fastest but the iterations are shown for the outer loop, i. e., for each outer iteration it performs 10 inner iteration steps and is therefore slower than the Conjugate Gradient method. In the next chapter preconditioning will be discussed and the speedup when solving instead of $Ax = b$ a nearby system $\tilde{A}x_0 = b$ that can be more easily solved and take x_0 as an approximation for x.

	Steepest Descent		GMRES(10)		CG	
Itr	p_m	ε_m	p_m	ε_m	p_m	ε_m
1	−0.002	$2.639 \cdot 10^{0}$	−0.035	$1.741 \cdot 10^{0}$	−0.003	$2.639 \cdot 10^{0}$
10	−0.016	$1.935 \cdot 10^{0}$	0.721	$2.949 \cdot 10^{-2}$	−0.052	$1.675 \cdot 10^{0}$
50	−0.055	$1.365 \cdot 10^{0}$	0.750	$1.071 \cdot 10^{-9}$	0.742	$1.839 \cdot 10^{-2}$
100	0.056	$1.037 \cdot 10^{0}$	–	–	0.750	$2.055 \cdot 10^{-7}$
500	0.593	$1.579 \cdot 10^{-1}$	–	–	0.750	$1.088 \cdot 10^{-14}$
1000	0.736	$1.410 \cdot 10^{-2}$	–	–	0.750	$1.088 \cdot 10^{-14}$

Table 4.2: Results of the projection methods for the model problem.

Chapter 5

Preconditioning

Although the methods seen in the previous chapter are well founded theoretically, they are likely to suffer from slow convergence for problems which arise from typical applications. Preconditioning is a key ingredient for the success of Krylov subspace methods in these applications. This chapter discusses preconditioning techniques and covers the methods successfully applied to the Conjugate Gradient method.

Starting with an introduction to preconditioning afterwards an optimized Cholesky factorization is presented to show a conventional but quite effective approach. Secondly, multigrid as preconditioner is introduced which is generally accepted as being the fastest numerical method for the solution of discretized elliptic partial differential equations.

5.1 Preconditioned Iterations

Lack of robustness is a widely recognized weakness of iterative solvers, relative to direct solvers. This drawback hampers the acceptance of iterative methods in industrial applications despite their intrinsic appeal for very large linear systems. Both the efficiency and robustness of iterative techniques can be improved by using preconditioning. The term preconditioning is simply a means of transforming the original linear system into one which has the same solution, but which is likely to be easier to solve with an iterative solver. In general, the reliability of iterative techniques, when dealing with various applications, depends much more on the quality of the preconditioner than on the particular Krylov subspace accelerators used.

Consider a matrix A that is symmetric and positive definite and assume that a preconditioner M is available. The preconditioner M is a matrix which approximates A in some yet-undefined sense. It is assumed that M is also symmetric positive definite. From a practical point of view, the only requirement of M is that it likely leads to an easier (computational more inexpensive) to solve linear systems $Mx = b$. This is because the preconditioned algorithms will require a linear system solution with the matrix M at each step. Then, for example, the following preconditioned system could be solved:

$$M^{-1}Ax = M^{-1}b$$

or

$$AM^{-1}u = b, \quad x = M^{-1}u.$$

Note that these two systems are no longer symmetric in general. To preserve symmetry a simple way is to "split"the preconditioner $M = LL^T$ performing a Cholesky factorization, which leads to

$$L^{-1}AL^{-T}u = L^{-1}b, \quad x = L^{-T}u.$$

However, it is not necessary to split the preconditioner in this manner in order to preserve symmetry. Observe that $M^{-1}A$ is self-adjoint for the M-inner product,

$$(x, y)_M \equiv (Mx, y) = (x, My)$$

since

$$(M^{-1}Ax, y)_M = (Ax, y) = (x, Ay) = (x, M(M^{-1}A)y) = (x, M^{-1}Ay)_M.$$

Therefore, an alternative is to replace the usual Euclidean inner product in the Conjugate Gradient Algorithm 8 by the M-inner product.

If the CG algorithm is rewritten for this new inner product, denoting by $r_k = b - Ax_k$ the original residual and by $z_k = M^{-1}r_k$ the residual for the preconditioned system, the following sequence of operations is obtained, ignoring the initial step:

1. $\alpha_k = (z_k, z_k)_M / (M^{-1}Ap_k, p_k)_M$
2. $x_{k+1} = x_k + \alpha_k p_k$
3. $r_{k+1} = r_k - \alpha_k Ap_k$ and $z_{k+1} = M^{-1}r_{k+1}$
4. $\beta_k = (z_{k+1}, z_{k+1})_M / (z_k, z_k)_M$
5. $p_{k+1} = z_{k+1} + \beta_k p_k$

Since $(z_k, z_k)_M = (r_k, z_k)$ and $(M^{-1}Ap_k, p_k)_M = (Ap_k, p_k)$, the M-inner products do not have to be computed explicitly. With this observation the preconditioned Conjugate Gradient Algorithm 9 is obtained.

Also note that in all the Krylov methods the individual elements of A do not have to be known and also it is never necessary to modify parts of the given matrix. It is always sufficient to have a rule (subroutine) that generates, for given input vector y, the output vector $z = Ay$. This also holds for the preconditioner: it does not have to be an explicitly given matrix. This property will be used later when using the multigrid method for preconditioning.

Algorithm 9 Preconditioned Conjugate Gradient algorithm.

```
$r_0 = b - Ax_0$, $z_0 = M^{-1}r_0$, $p_0 = z_0$
Until convergence do $k = k + 1$:
    $\alpha_k = (r_k, z_k)/(Ap_k, p_k)$
    $x^{(k+1)} = x^{(k)} + \alpha_k p_k$
    $r_{k+1} = r_k - \alpha_k A p_k$
    $z_{k+1} = M^{-1} r_{k+1}$
    $\beta_k = (r_{k+1}, z_{k+1})/(r_k, z_k)$
    $p_{k+1} = z_{k+1} + \beta_k p_k$
```

5.2 Cholesky Factorization

For a given symmetric positive definite matrix $A \in \mathbb{R}^{n \times n}$ there exists a unique lower triangular $L \in \mathbb{R}^{n \times n}$ with positive diagonal entries such that $A = LL^T$. This factorization is known as Cholesky factorization and L is known as Cholesky triangle. If A is tridiagonal then L has just one subdiagonal and this property can be used to construct to build a more efficient factorization: only $4n$ instead of $n^3/3$ operations. Extending this idea to a block tri-diagonal matrix with blocksize s it leads to a cost of $\frac{7}{3}ns^3$ operations (Cao et al. [12]). Algorithm 10 shows a block tri-diagonal Cholesky factorization with variables as described in Equation 5.1.

Algorithm 10 Block Tri-Diagonal Cholesky Factorization.

```
$D_1 = \mathrm{chol}(A_{11})$
For $i = 1, 2, \ldots, \lceil m/s \rceil - 1$ do:
    $E_i = D_i^{-T} A_{i,i+1}$
    $D_{i+1} = \mathrm{chol}(A_{i+1,i+1} - E_i^T E_i)$
```

$$A = \begin{pmatrix} A_{11} & A_{12} & \\ A_{12} & A_{22} & \ddots \\ & \ddots & \ddots \end{pmatrix} \qquad \mathrm{chol}(A) = \begin{pmatrix} D_1 & E_1 & \\ & D_2 & E_2 \\ & & \ddots \end{pmatrix} \tag{5.1}$$

The basic block tri-diagonal Cholesky factorization can be further enhanced. Using the incomplete Cholesky factorization (Saad [47]) lowers the amount of fill in and experiments have shown that a threshold value of 10^{-3} leads to best results, i. e., the preconditioning matrix contain only values larger than 10^{-3}. Another performance gain can be achieved by using a bandwidth reduction algorithm prior to the factorization as suggested in Chapter 3.2. Table 5.1 shows the different results for not preconditioned Conjugate Gradient method, preconditioning with the standard variant as is Algorithm 5.1 and the optimized version with incomplete Cholesky factorization and the reverse Cuthill McKey bandwidth reduction algorithm—p_m is the value at the center of the model (4.3) and ε_m the maximum absolute error (4.4).

	CG without precond.		CG with std. btdchol		CG with opt. btdchol	
Itr	p_m	ε_m	p_m	ε_m	p_m	ε_m
1	-0.003	$2.639 \cdot 10^{0}$	-0.040	$1.267 \cdot 10^{0}$	-0.042	$1.292 \cdot 10^{0}$
5	-0.015	$2.079 \cdot 10^{0}$	0.758	$1.118 \cdot 10^{-2}$	0.754	$5.924 \cdot 10^{-3}$
10	-0.052	$1.676 \cdot 10^{0}$	0.750	$3.366 \cdot 10^{-6}$	0.750	$4.905 \cdot 10^{-7}$
15	-0.096	$1.346 \cdot 10^{0}$	0.750	$4.289 \cdot 10^{-10}$	0.750	$4.360 \cdot 10^{-11}$
20	-0.147	$1.095 \cdot 10^{0}$	0.750	$4.929 \cdot 10^{-14}$	0.750	$5.107 \cdot 10^{-15}$
25	-0.115	$8.655 \cdot 10^{-1}$	0.750	$2.655 \cdot 10^{-15}$	0.750	$2.665 \cdot 10^{-15}$

Table 5.1: Conjugate Gradient method with different preconditioning strategies.

Results for the actual systems with a bandwidth of about 8 000 after applying a bandwidth reduction algorithm show that runtime performance is still bad because Cholesky factorization of each block involves $O(n^3)$ flops which slows down the complete algorithm and nullifies the speed up of the preconditioned CG method. A natural choice would be therefore to use in step 1 and 4 of the block tridiagonal Cholesky factorization this algorithm recursively. Although those blocks are still sparse they don't have a low bandwidth anymore and would require again a bandwidth reduction, i. e., a row/column permutation that would also effect the E_i blocks (see Equation 5.1). To overcome this problem techniques known from parallel Cholesky factorizations could be used where the matrix is partitioned in a way, that reordering one block does not effect another block—see Cao et al. [12]. But with additional complexity the gain of applying a preconditioner gets smaller and the most important conclusion that can be drawn from this section is the importance and speed up of preconditioners for the CG method. The next section will show another type of preconditioner which also can handle larger sizes efficiently.

5.3 Multigrid Methods

Multigrid methods were first described in the 1960s and their actual efficiency was recognized by Brandt in the early 1970s. Since this time those methods became more and more important and now are regarded to be among the fastest methods for many problems like all kinds of partial differential equations, integral equations etc. This section is based on *Multigrid* by Trottenberg et al. [53] and mainly follows the structure of this book.

At first the multigrid idea is introduced heuristically for the model problem (4.1) of Chapter 4. If the lexicographical Gauss-Seidel method (Algorithm 3) is applied to Poisson's equation, the following phenomenon is observed. After a few iteration steps, the error of the approximation becomes smooth. It doesn't necessarily become small, but it does become smooth—see Fig. 5.1 for an illustration of this effect.

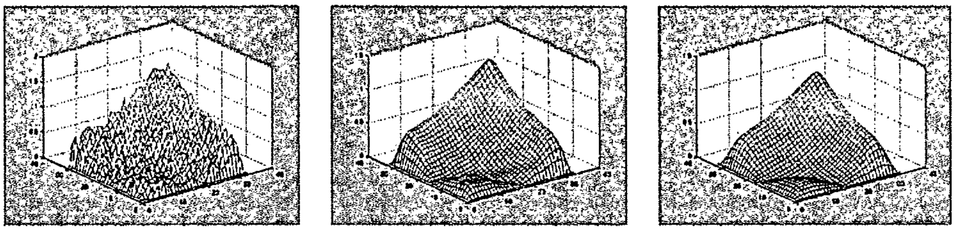


Figure 5.1: Influence of lexicographic Gauss-Seidel iteration on the error for the model problem after 1, 5 and 10 iteration steps (from left to right).

Obviously, the iteration formula can be interpreted as an error averaging process. Error smoothing is one of the two basic principles of the multigrid approach.

The other basic principle is the following: a quantity that is smooth on a certain grid can, without any essential loss of information, also be approximated on a coarser grid, i. e., a grid with double the mesh size. Qualitatively, this is the coarse grid approximation principle and as this principle holds for error or "correction" quantities, it is also called the coarse grid correction (CGC) principle.

The idea of multigrid can therefore be summarized into:

Smoothing principle Many classical iterative methods (Gauss-Seidel, etc.) if appropriately applied to discrete elliptic problems have a strong smoothing effect on the error of any approximation.

Coarse Grid principle A smooth error term is well approximated on a coarse grid. A coarse grid procedure is substantially less expensive (substantially fewer grid points) than a fine grid procedure.

These considerations can be explained heuristically by looking at the Fourier expansion of the error and studying the high and low frequency components. (The local Fourier analysis is the most powerful tool for the quantitative analysis and the design of efficient multigrid methods for general problems—ee Brandt [10]).

5.3.1 Two-grid cycle

We start by introducing the two-grid cycle, the natural basis for any multigrid algorithm. For this purpose a discrete linear elliptic boundary value problem of the form

$$L_h u_h = f_h \quad (\Omega_h) \tag{5.2}$$

is used and we assume L_h^{-1} exists. As example for performance evaluation we again use the model problem (4.1) from Chapter 4.

For any approximation u_h^m of the solution u_h of (5.2), we denote the error by

$$v_h^m = u_h - u_h^m$$

and the defect (or residual) by

$$d_h^m = f_h - L_h u_h^m.$$

Trivially, the defect equation $L_h v_h^m = d_h^m$ is equivalent to the original equation (5.2) since $u_h = u_h^m + v_h^m$. If L_h in the defect equation is now approximated by any "simpler"operator $\hat{L}_h$ such that $\hat{L}_h^{-1}$ exists, the solution $\hat{v}_h^m$ of

$$\hat{L}_h \hat{v}_h^m = d_h^m$$

gives a new approximation

$$u_h^{m+1} = u_h^m + \hat{v}_h^m.$$

The procedural formulation then looks like

$$u_h^m \quad \rightsquigarrow \quad d_h^m = f_h - L_h u_h^m \quad \rightsquigarrow \quad \hat{L}_h \hat{v}_h^m = d_h^m \quad \rightsquigarrow \quad u_h^{m+1} = u_h^m + \hat{v}_h^m.$$

One idea to approximately solve the defect equation is to use an appropriate approximation L_H of L_h on a coarser grid Ω_H, for instance the grid with mesh size $H = 2h$. We assume two (linear) transfer operators

$$I_h^H : \mathcal{G}(\Omega_h) \to \mathcal{G}(\Omega_H), \qquad I_H^h : \mathcal{G}(\Omega_H) \to \mathcal{G}(\Omega_h)$$

to be given. I_h^H is used to restrict d_h^m to Ω_H and I_H^h is used to interpolate (or prolongate) the correction $\hat{v}_H^m$ to Ω_h.

Combining the above considerations of the two processes of smoothing and of coarse grid correction leads to the two-grid method summarized in Algorithm 11.

Algorithm 11 Two-grid cycle.

```
1. Presmoothing:
     Compute $\bar{u}_h^m$ by applying $\nu_1$ ($\geq 0$) steps of a given smoothing procedure to $u_h^m$
2. Coarse grid correction:
     Compute the defect                              $\bar{d}_h^m = f_h - L_h \bar{u}_h^m$
     Restrict the defect                             $\bar{d}_H^m = I_h^H \bar{d}_h^m$
     Solve on $\Omega_H$                             $L_H \hat{v}_H^m = \bar{d}_H^m$
     Interpolate the correction                      $\hat{v}_h^m = I_H^h \hat{v}_H^m$
     Compute the the corrected approximation         $u_h^{m,\text{after CGC}} = \bar{u}_h^m + \hat{v}_h^m$
3. Postsmoothing
     Compute $u_h^{m+1}$ by applying $\nu_2$ ($\geq 0$) steps of the given smoothing procedure to
     $u_h^{m,\text{after CGC}}$
```

5.3.2 Components of Multigrid

Error smoothing

Classical iteration methods such as Gauss-Seidel-type and Jacobi-type iterations are often called relaxation methods (also smoothing methods or smoothers) if they are used for the purpose of error smoothing. In general, however, appropriate Gauss-Seidel-type iterations turn out to be better smoothers than appropriate Jacobi-type iterations so we will focus on Gauss-Seidel here. In Section 3.1.1 the lexicographic ordering of grid points (GS-LEX) was introduced. A different ordering is the so-called red-black ordering (GS-RB)—see Fig. 5.2.

We also recall that for the model problem the convergence of Gauss-Seidel iteration can be substantially improved by an overrelaxation parameter

$$\omega = \frac{2}{1 + \sin \pi h}.$$

This is the classical result on successive overrelaxation (SOR) and was described in the multigrid context by Yavneh [60].

Gauss-Seidel-type methods represent a particularly important class of smoothers. In the multigrid context the smoothing properties of Gauss-Seidel are much more important than the convergence properties. In order to measure the smoothing properties quantitatively a smoothing factor μ is used and it represents the worst factor by which high frequency error components are reduced per relaxation step.

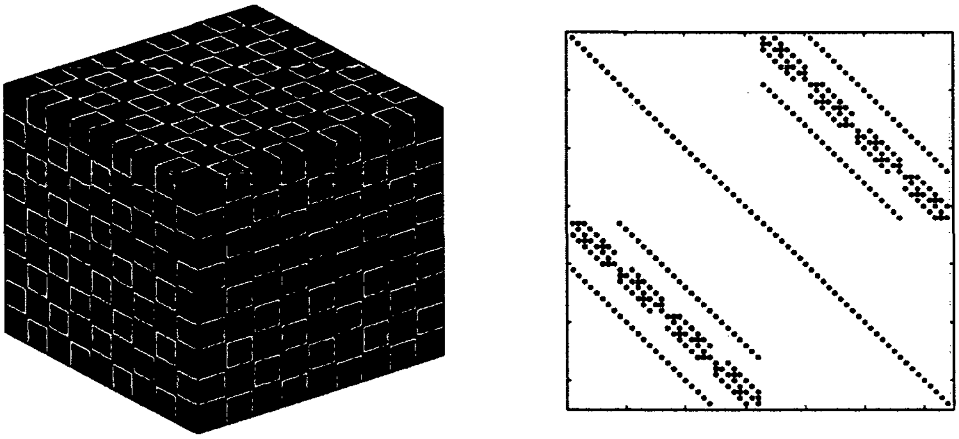


Figure 5.2: A three dimensional checkerboard and the corresponding sparsity pattern of the associated matrix.

Trottenberg et al. [53] obtain for the 3D case

$$\begin{aligned}\mu(\text{GS-LEX}) &= 0.445 &&(\text{for } \omega = 1)\\ \mu(\text{GS-RB}) &= 0.567 &&(\text{for } \omega = 1)\\ \mu(\text{GS-RB}) &= 0.23 &&(\text{for } \omega = 1.15).\end{aligned}$$

Choices for the coarse grid

The simplest and most frequently used choice for the grid Ω_H is standard coarsening, doubling the mesh size h in every direction. In d dimensions, the relation between the number of grid points (neglecting boundary effects) is

$$\#\Omega_H \approx \frac{1}{2^d}\#\Omega_h.$$

We speak of semicoarsening if the mesh size h is doubled in less than d directions only, which is especially of interest for anisotropic operators (Washio and Oosterlee [58]). In the context of the AMG approach (see below), the coarse grid Ω_H is not formed according to such a fixed simple strategy. Using the algebraic relations in the corresponding matrix, Ω_H is determined by the AMG process itself in the course of calculation.

However, for further investigations (except AMG) we assume that $h_{k+1}/h_k = 1/2$ (with the mesh size h_i for the corresponding grid G^i) since this ratio of mesh size is the most efficient (Brandt [10]).

Restriction and Prolongation operator

The choice of restriction and interpolation operators I_h^H and I_H^h, for the intergrid transfer of grid functions, is closely related to the choice of the coarse grid. A restriction operator I_h^{2h} maps h-grid function to $2h$-grid functions. The simplest example is the injection operator, which identifies grid functions at coarse grid points with the corresponding grid functions at fine grid points. Hackbusch [27] although discourages the use of the injection operator.

Another frequently used restriction operator is the full weighting (FW) operator, which in the 3D case and stencil notation reads

$$\frac{1}{64}\left[\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}_h^{2h}\begin{bmatrix}2&4&2\\4&8&4\\2&4&2\end{bmatrix}_h^{2h}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}_h^{2h}\right].$$

The prolongation (interpolation) operators map $2h$-grid functions into h-grid functions. A frequently used interpolation method in 3D is trilinear interpolation from G^{2h} to G^h, which is given by

$$I_{2h}^h\hat{v}_{2h}(x,y,z)=\begin{cases}\hat{v}_{2h}(x,y,z) & \text{for } \bullet\\ \frac{1}{2}[\hat{v}_{2h}(x+h,y,z)+\hat{v}_{2h}(x-h,y,z) & \text{for } \square\\ \frac{1}{2}[\hat{v}_{2h}(x,y+h,z)+\hat{v}_{2h}(x,y-h,z) & \text{for } \boxplus\\ \frac{1}{2}[\hat{v}_{2h}(x,y,z+h)+\hat{v}_{2h}(x,y,z-h) & \text{for } \boxtimes\\ \frac{1}{4}[\hat{v}_{2h}(x+h,y+h,z)+\hat{v}_{2h}(x+h,y-h,z) & \\ \quad+\hat{v}_{2h}(x-h,y+h,z)+\hat{v}_{2h}(x-h,y-h,z) & \text{for } \circ\\ \frac{1}{4}[\hat{v}_{2h}(x+h,y,z+h)+\hat{v}_{2h}(x-h,y,z-h) & \\ \quad+\hat{v}_{2h}(x-h,y,z+h)+\hat{v}_{2h}(x-h,y,z-h) & \text{for } \oplus\\ \frac{1}{4}[\hat{v}_{2h}(x,y+h,z+h)+\hat{v}_{2h}(x,y-h,z-h) & \\ \quad+\hat{v}_{2h}(x,y+h,z-h)+\hat{v}_{2h}(x,y-h,z-h) & \text{for } \otimes\\ \frac{1}{8}[\hat{v}_{2h}(x+h,y+h,z+h)+\hat{v}_{2h}(x+h,y+h,z-h) & \\ \quad+\hat{v}_{2h}(x+h,y-h,z+h)+\hat{v}_{2h}(x+h,y-h,z-h) & \\ \quad+\hat{v}_{2h}(x-h,y+h,z+h)+\hat{v}_{2h}(x-h,y+h,z-h) & \\ \quad+\hat{v}_{2h}(x-h,y-h,z+h)+\hat{v}_{2h}(x-h,y-h,z-h) & \text{for } \triangle\end{cases}\tag{5.3}$$

Fig. 5.3 presents (part of) a fine grid with the symbols for the fine and coarse grid points referred to by (5.3).

It can be shown that the interpolation operator corresponds to the FW restriction operator in a natural way: these two operators are transpose to each other (see Hackbusch [26])

$$I_H^h=\left(I_h^H\right)^T.$$

5.3.3 Multigrid and Full Multigrid Cycle

The previous section described the multigrid principle only in its two-grid version. The multigrid idea starts from the observation that in a well converged two-grid method it is neither useful nor necessary to solve the coarse grid defect equation

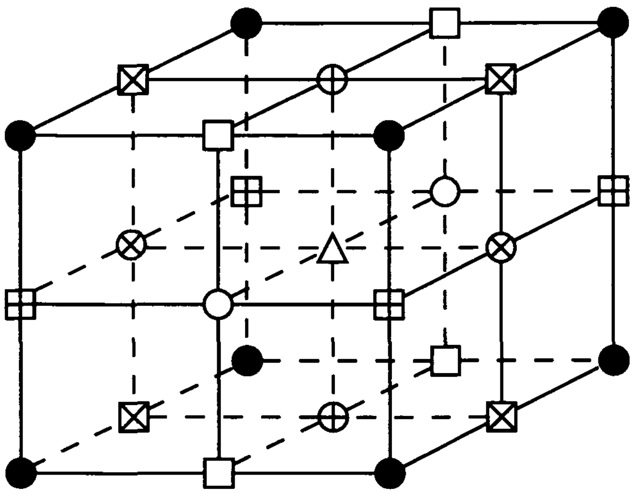


Figure 5.3: A fine grid with symbols indicating the trilinear interpolation 5.3 used for the transfer from the coarse gird (•).

exactly. Instead, without essential loss of convergence speed, one may replace $\hat{v}_H^m$ by a suitable approximation. A natural way to obtain such an approximation is to apply the two-grid idea again, now employing an even coarser grid than Ω_H. This is possible, as obviously the coarse grid equation is of the same form as the original equation. If convergence of the two-grid method is fast enough, it is sufficient to perform only a few two-grid iteration steps (see Fig. 5.4) to obtain a good enough approximation to the solution. This idea can, in a straightforward manner, be applied recursively, using coarser and coarser grids, down to some coarsest grid. On this coarsest grid any solution method may be used (e. g., a direct method or some relaxation-type method if it has sufficiently good convergence properties on that coarsest grid).

For obvious reasons, we refer to the case $\gamma = 1$ as V-cycles and to $\gamma = 2$ as W-cycles. Usually, the cases $\gamma = 1$ and $\gamma = 2$ are particularly interesting, but the cycle index γ need not to be a fixed number. The so-called F-cycle which is illustrated in Fig. 5.5 is an example for a variable cycle index.

Table 5.2 compares the results for different cycle indices. We test again with the model problem (4.1) from Chapter 4 but with $h = 1/34$ ($\rightsquigarrow$ dim=35 937) using 4 grids. This leads to dim=125 on the coarsest grid and is solved there directly. As the results show it is not necessary for the model problem to use the computational more expensive W- or F-cycle. Results for a more complex problem are given in Chapter 7.

An initial approximation for iterative solvers, like multigrid, can be obtained by nested iteration. The general idea of nested iteration is to provide an initial approximation on a grid Ω_l by the computation and interpolation of approximations

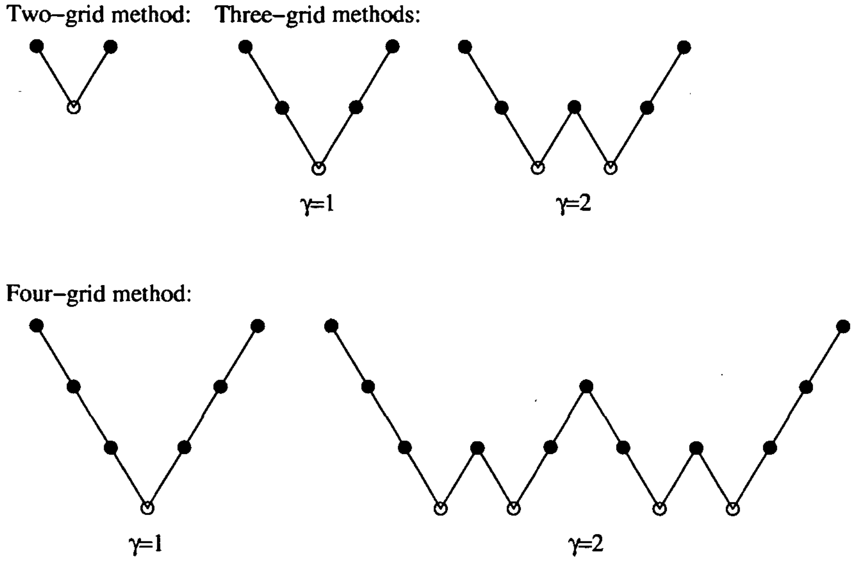


Figure 5.4: Structure of one multigrid cycle for different number of grids and different values of the cycle index γ (• smoothing, ∘ exact solution, \ fine-to-coarse, / coarse-to-fine transfer).

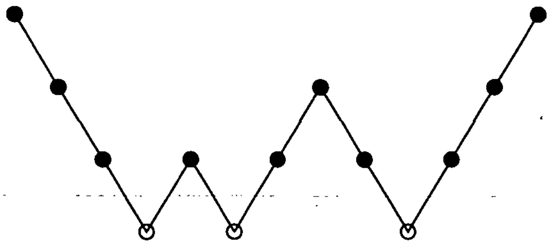


Figure 5.5: Structure of an F-cycle.

	$\gamma = 1$		$\gamma = 2$		F-cylcle	
Itr	p_m	ε_m	p_m	ε_m	p_m	ε_m
1	0.758	$1.340 \cdot 10^{-1}$	0.755	$1.287 \cdot 10^{-1}$	0.765	$1.285 \cdot 10^{-1}$
2	0.751	$3.494 \cdot 10^{-3}$	0.750	$3.024 \cdot 10^{-3}$	0.750	$3.023 \cdot 10^{-3}$
3	0.750	$1.280 \cdot 10^{-4}$	0.750	$1.614 \cdot 10^{-4}$	0.750	$1.617 \cdot 10^{-4}$
4	0.750	$5.126 \cdot 10^{-6}$	0.750	$6.416 \cdot 10^{-6}$	0.750	$6.424 \cdot 10^{-6}$
5	0.750	$2.211 \cdot 10^{-7}$	0.750	$2.756 \cdot 10^{-7}$	0.750	$2.760 \cdot 10^{-7}$

Table 5.2: Results of the multigrid method with varying cycle index for the model problem.

on coarser girds. Within an arbitrary iterative process for the solution of a given discrete problem, this principle simply means that a lower (coarser) discretization level is used in order to provide a good initial approximation for the iteration on the next higher (finer) discretization level.

The efficiency of iterative multigrid can be improved if it is properly combined with the nested iteration idea. This combination is called the full multigrid (FMG) technique (Brandt [10]). Typically, the FMG scheme is the most efficient multigrid version and is illustrated in Fig. 5.6.

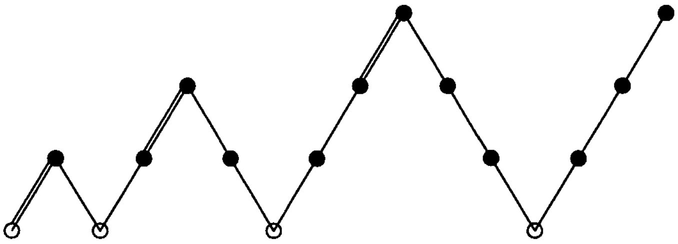


Figure 5.6: Structure of an FMG-cycle with the double line symbolizing the FMG interpolation.

It is not sufficient to start the solution process on a very coarse grid, interpolate the approximation of the coarse grid solution to the next finer grid, smooth the visible error components and so on until the finest grid is reached. Actually, the interpolation of the approximation leads to nonnegligible high and low frequency error components on the fine grid that can efficiently be reduced only by subsequent smoothing of the error on all grid levels, i. e., by revisiting the coarse levels in multigrid cycles.

Table 5.3 shows FMG with a different number of grid levels for the model problem. As expected two levels of grids perform bad and the result for the 4-level grid is somewhere between the first and second iteration of a standard iterative multigrid method.

	FMG	
# grids	p_m	ε_m
2	1.506	$2.379 \cdot 10^{0}$
3	0.735	$4.193 \cdot 10^{-2}$
4	0.747	$4.401 \cdot 10^{-2}$

Table 5.3: Results of FMG with a different number of grid levels.

5.3.4 Algebraic Multigrid

A natural question arises: Can multigrid techniques be applied when there is no grid? That is, when there is a relationship among the unknowns, but the physical locations of the unknowns are themselves unknown or immaterial. These are the problems that are addressed by a technique known as algebraic multigrid, or AMG. For any multigrid algorithm, the same fundamental components are required. There must be a sequence of grids, an intergrid transfer operator, a relaxation (smoothing) operator, coarse-grid versions of the fine-grid operator, and a solver for the coarsest grid.

In particular, coarse-grid discretizations used in geometric multigrid to reduce low-frequency error components now correspond to certain matrix equations of reduced dimension. However, no multigrid hierarchy needs to be known a priori. In fact, the construction of a (problem-dependent) hierarchy is part of the AMG algorithm, based solely on algebraic information contained in the given system of equations.

The geometric approach employs fixed grid hierarchies and, therefore, an efficient interplay between smoothing and coarse-grid correction has to be ensured by selecting appropriate smoothing processes. In contrast to this, AMG fixes the smoother to some simple relaxation scheme such as plain Gauss-Seidel relaxation, and enforces an efficient interplay with the coarse-grid correction by choosing the coarser levels and interpolation appropriately. Geometrically speaking, AMG attempts to coarsen only in directions in which relaxation really smoothes the error for the problem at hand. However, since the relevant information is contained in the matrix itself (in terms of size and signs of coefficients), this process can be performed based only on matrix information, producing coarser levels which are locally adapted to the smoothing properties of the given smoother.

The coarsening process is fully automatic. This automation is the major reason for AMG's flexibility in adapting itself to specific requirements of the problems to be solved and is the main reason for its robustness in solving large classes of problems despite using very simple pointwise smoothers. But the flexibility of AMG and its simplicity of use, of course, have a price: A setup phase, in which the given problem is analyzed, the coarse levels are constructed and all operators are assembled, has to be concluded before the actual solution phase can start. This extra overhead is one reason why AMG is usually less efficient than geometric multigrid approaches (if applied to problems for which geometric multigrid can be applied efficiently).

It is not an easy task to implement all the components necessary for AMG from scratch. Therefore, a tool developed at the Institute of Computational Mathematics at the Johannes Kepler University of Linz in Austria, named PEBBLES (see Section 6.1), was used. Table 5.4 shows the results of PEBBLES solving the model problem (4.1) with different coarsening strategies (strong—Ruge and Stüben [46], agglomeration—Kickinger [32], vmb—Vanek et al. [54]). The configuration was choosen to use again 4 levels of grid hierarchies ($\rightsquigarrow$ dim=384, dim=235, dim=549 for the respective coarsening strategies), a V-cycle and cholesky factorization as coarse grid solver.

	strong		agglomeration		vmb	
Itr	p_m	ε_m	p_m	ε_m	p_m	ε_m
1	0.687	$2.634 \cdot 10^{-1}$	0.654	$3.474 \cdot 10^{-1}$	0.726	$2.555 \cdot 10^{-1}$
2	0.745	$3.560 \cdot 10^{-2}$	0.739	$7.818 \cdot 10^{-2}$	0.751	$3.256 \cdot 10^{-2}$
3	0.750	$5.006 \cdot 10^{-3}$	0.749	$1.922 \cdot 10^{-2}$	0.750	$4.411 \cdot 10^{-3}$
4	0.750	$6.357 \cdot 10^{-4}$	0.750	$4.936 \cdot 10^{-3}$	0.750	$5.112 \cdot 10^{-4}$
5	0.750	$1.654 \cdot 10^{-4}$	0.750	$1.285 \cdot 10^{-3}$	0.750	$1.654 \cdot 10^{-4}$

Table 5.4: Results of algebraic multigrid with various coarsening strategies.

5.3.5 Multigrid as preconditioner

In order to increase the robustness of standard multigrid approaches, it has become very popular in recent years to use multigrid not as stand-alone solver but rather to combine it with acceleration methods such as conjugate gradient, BI-CGSTAB or GMRES. This development was driven by the observation that is is often not only simpler but also more efficient to use accelerated multigrid approaches rather than to try to optimize the interplay betweenthe various multigrid components in order to improve the convergence of stand-alone multigrid cycles.

In the simplest case, complete multigrid cycles are merely used as preconditioners when solving step 6 in Algorithm 9

$$z_k = M^{-1} r_k.$$

In more sophisticated approaches, acceleration is even used on the individual grids of the hierarchy. E. g., Bank and Douglas [7] treated the conjugate gradient method as a relaxation method of the multigrid method.

For AMG which was originally designed to be used stand-alone it has turned out that the situation is quite similar to standard multigrid. Practical experience has clearly shown that AMG is also a very good preconditioner, much better than standard (one-level) ILU-type preconditioners, for example. Heuristically, the major reason is due to the fact that AMG, in contrast to any one-level preconditioner, operates efficiently an all error components. This has the implication that, instead of using AMG stand-alone, it is generally more efficient to put less effort into the (expensive) setup phase and use AMG as preconditioner, for example, by using aggressive coarsening strategies.

So the question whether multigrid should be used as solver or as a preconditioner is in particular the question which approach should be used when. It is, of course, not useful to accelerate a highly efficient multigrid algorithm by Krylov subspace acceleration because the extra effort does not pay off. An argument for combining multigrid with an acceleration technique is that problems become more and more complex if we treat real-life applications. For such complicated applications like FES, it is far from trivial to choose optimal multigrid components. Therefore, when certain error components may remain large since they cannot be reduced by standard smoothing procedures combined with standard coarse grid approximations, Krylov subspace methods may have the potential of a substantial acceleration.

Table 5.5 shows the results for the model problem when multigrid or algebraic multigrid is used as preconditioner. Since the Poisson Equation does not show any anisotropies, discontinuities and the like the results are almost the same as with pure multigrid. In Chapter 7 we will see a different situation when applying the above algorithms to a more complex model problem and a real-life application.

	MG		AMG	
Itr	p_m	ε_m	p_m	ε_m
1	0.772	$9.033 \cdot 10^{-2}$	0.737	$1.370 \cdot 10^{-1}$
2	0.751	$2.695 \cdot 10^{-3}$	0.752	$7.625 \cdot 10^{-3}$
3	0.750	$6.322 \cdot 10^{-5}$	0.750	$3.422 \cdot 10^{-4}$
4	0.750	$5.462 \cdot 10^{-6}$	0.750	$1.722 \cdot 10^{-4}$
5	0.750	$2.065 \cdot 10^{-7}$	0.750	$1.654 \cdot 10^{-4}$

Table 5.5: Results of multigrid as preconditioner for the conjugate gradient algorithm.

Chapter 6

Implementation

This chapter addresses implementation issues of the algorithms described in the previous chapters. The quality of the computer implementation of sparse matrix algorithms can have a profound impact on their performance, and the difficulty of implementation varies a great deal from one algorithm to another. Thus, while "paper and pencil" analysis of sparse matrix algorithms are useful and important, they are not enough (George and Liu [23]).

In particular this chapter describes the implementation of an AMG software package and two systems for remote and distributed computing to overcome the limitations of executing the solver on a local Windows machine. The program code presented in this section is available on the included CD as well as online at the following location:
`http://fsmat.at/~cfabiane/docs/fes_solver.tgz`

6.1 PEBBLES

PEBBLES is the acronym for Parallel and Element Based grey Box Linear Equation Solver. This sophisticated program package was developed at the Institute of Computational Mathematics at the Johannes Kepler University of Linz, Austria in the course of the research project SFB F013 'Numerical and Symbolic Scientific Computing' funded by the Austrian Science Foundation (FWF). At first we give a short introduction into the field of application as described in the software manual (Reitzinger [45]) followed by the description of how to port PEBBLES to other architectures to meet the requirements for the FES-Tool at the Department of Biomedical Engineering and Physics.

6.1.1 Field of Application

The intention of PEBBLES is to provide a fast and robust iterative solver for large, sparse, (symmetric) positive definite (spd) matrices which arise typically in an Finite Element discretization of second order elliptic problems. PEBBLES solves

$$K_h u_h = f_h \tag{6.1}$$

where $K_h \in \mathbb{R}^{N_h \times N_h}$ is a (s)pd stiffness matrix, $f_h \in \mathbb{R}^{N_h}$ is the right hand side and $u_h \in \mathbb{R}^{N_h}$ is the solution vector. Moreover, K_h is assumed to arise from an FE-discretization of an elliptic partial differential equation of second order. $N_h = O(h^{-d})$ (where $d = 2, 3$ is the spatial dimension) is related to the number of unknowns and h the discretization parameter. PEBBLES bases essentially on algebraic multigrid, so no hierarchical grid structure is required.

PEBBLES is intended to solve the following problems.

- Equations of the form (6.1), which have to be a discretization of an elliptic partial differential equation of second order.
- Scalar problems as well as block structured problems if the system matrix is spd, if the discretization was done by nodal FE-functions and if there are no anisotropies in the problem or the discretization. The last point is actually only necessary for block structured problems.
- Solve edge FE-discretizations.
- Make an element preconditioning if the problem arises from a selfadjoint, scalar problem.
- Moderately non-symmetric problems can be handled through the BiCGStab method.

To configure PEBBLES a so called navigator file (`pebbles.inp`) is used. There it is possible to define which solution strategy, accuracy and other parameters should be used.

6.1.2 Porting PEBBLES to Other Architectures

PEBBLES is natively available for the following operating systems / architectures: Linux, Solaris, and SGI Origin. Since the FES-Tool (MATLAB) and the whole working environment at the Department of Biomedical Engineering and Physics is under Windows in a first step we built an interface to access PEBBLES from within MATLAB and then port it to Windows using Cygwin [1].

MATLAB provides an Application Program Interface (API) to support external interfaces to programs written in C or Fortran [40]. The API allows (amongst other sophisticated features) calling C programs and importing / exporting data to and from the MATLAB environment. MATLAB callable C and Fortran programs are referred to as MEX-files, which are dynamically linked subroutines that the MATLAB interpreter can automatically load and execute. This interface was used as a wrapper around PEBBLES and since MATLAB and PEBBLES use the same storage format for sparse matrices almost the same performance as calling PEBBLES natively was achieved. In order to use all the configuration options of the navigator file (`pebbles.inp`) the code was slightly changed and these options can be passed as a semicolon separated string of options. A typicall call of PEBBLES from within MATLAB now looks like:

```
>> x=pebbles(A,b,x,'EPS_PCG=1e-8;ELEMENT_PRECOND=0');
```

The source files for building the wrapper are in `fes_solver.tgz`, subdirectory `matlab_helper`.

After building an interface to MATLAB the next step was to port PEBBLES to the Windows operating system. As mentioned above Cygwin was used which is a Linux-like environment for Windows consisting of a DLL (`cygwin1.dll`) which acts as a Linux emulation layer providing substantial Linux API functionality and a collection of tools, which provide Linux look and feel. With the help of Cygwin and a few changes it was possible to compile PEBBLES and build a DLL which the MEX compiler of MATLAB could use. Those changes and a step by step instruction how this can be accomplished is again found in `fes_solver.tgz` in the subdirectory `win_helper`.

As will be seen in Chapter 7 PEBBLES will crash if the problem size exceeds a certain dimension because of memory limits. Therefore, we successfully compiled PEBBLES on an HP Itanium 64bit system and were able to process even the largest matrices. The necessary makefile for PEBBLES is in `fes_solver.tgz`, subdirectory `ia64`.

6.2 NetSolve

NetSolve is a project whose goal is to provide easy, uniform, and efficient access to a wide variety of numerical software distributed over a network. NetSolve also strives to overcome some of the problems inherent in using those software libraries. One such problem is that in order to use some libraries, the user must have a high level of programming proficiency. In addition, some libraries may not be optimized for all platforms or may not be available at all on some systems.

One way in which NetSolve attempts to overcome these obstacles is to provide many interfaces, including C, Fortran and MATLAB, interfaces. Providing many interfaces not only widens the range of systems NetSolve is applicable to, but it also widens the range of users who could utilize NetSolve. For example, a scientist may have a machine capable of running the C interface to NetSolve, but if he is not proficient with C programming, it is doubtful he would have an interest in NetSolve. However, having access to simpler interfaces like the MATLAB interface, which do not require the user to do any programming, may interest such a user more than the programming-based interfaces.

This section presents an overview of the NetSolve system, including a description of its available interfaces and its approach to load balancing and fault tolerance.

6.2.1 The NetSolve System

NetSolve is a client-server application consisting of a possibly heterogeneous set of machines connected over a network. Each machine is running at least one NetSolve daemon, of which there are two kinds: the "computational server" and the "agent". The computational server is the process that has access to the mathematical libraries on that machine and solves problems on behalf of the client. The agent is an entity that keeps track of the status of all machines in the NetSolve system and determines which computational server would be best suited to handle each request. Figure 6.1 shows a typical NetSolve system.

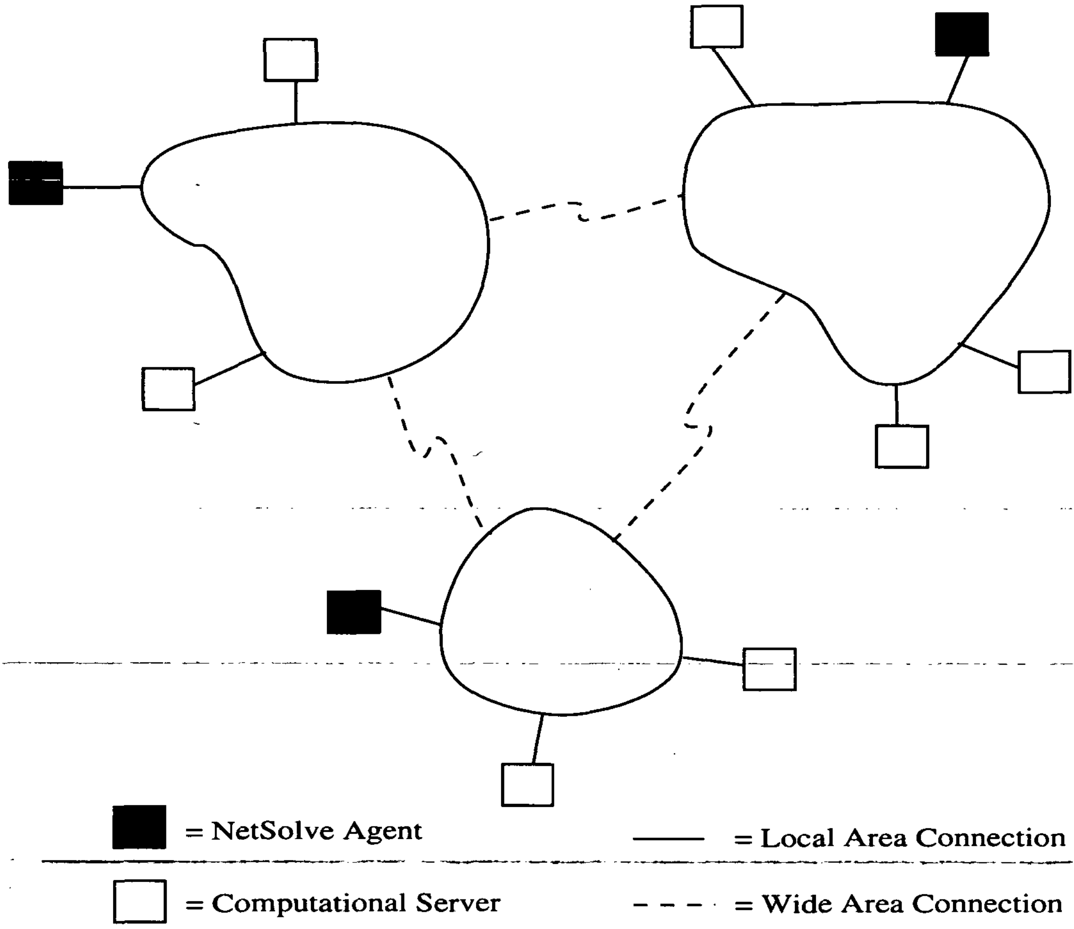


Figure 6.1: A Typical NetSolve System.

Each agent maintains its own information on the configuration of the NetSolve System which may differ from the information on other agents, but once the system becomes stable, all agents should converge to the same view of the system.

NetSolve is designed so that these servers may be added or removed any time without adversely affecting the whole system. To aid in modifying the NetSolve system, NetSolve provides interactive management utility programs which provide information and statistics about the current NetSolve system and allow dynamic modification of the system.

6.2.2 Interfaces

As stated earlier, one of the goals of NetSolve is to provide many interfaces through which users may access computational resources. Currently the available interfaces may be categorized into programming interfaces and non-programming (interactive) interfaces. The programming interfaces (C and Fortran) require the user to write, compile, and execute code in order to solve a problem. The non-programming interfaces (MATLAB, Mathematica and Octave) allow the user to solve problems without writing any code at all.

Although some users may not wish to learn programming, there are many users who are already familiar with programming and, in fact, have already written programs that utilize mathematical libraries. The programming interfaces were designed so that such programs could be modified to access these libraries through NetSolve as easily as possible. For example, the original Fortran program may contain a call to LAPACK's [2] `dgesv()` as follows:

```
call DGESV(N,1,A,MAX,IPIV,B,MAX,INFO)
```

The equivalent call using NetSolve would be:

```
call NETSL('DGESV()',NSINFO,
           N,1,A,MAX,IPIV,B,MAX,INFO)
```

As the preceding example shows, it takes very little programming effort to convert a standard library function call to a NetSolve function call. Even writing a new program that uses NetSolve to access a given mathematical library is no harder than writing a new program that uses that library directly. Also, there is a possible benefit from calling such functions via NetSolve. NetSolve may be called asynchronously so that further computations can take place on the client machine while the NetSolve computational server is fulfilling the request. Sending the requests asynchronously can even provide some parallelism provided there are multiple machines with access to the library the user needs to call. The user would just send multiple requests asynchronously and the NetSolve agent would use its load balancing scheme to distribute the requests among the available computational servers such that the requests are fulfilled in the least amount of time. See Arnold et al. [3] for a detailed description of the load balancing scheme used by NetSolve and how it can provide parallelism.

The programming interfaces were designed with ease of use in mind but some users may still choose to avoid programming for many reasons, including the difficulty of learning the language. However, even users with a thorough knowledge of programming may prefer an interactive interface for some problems because the time involved in writing a computer program may be prohibitive, even for a simple task. For instance, let us suppose a user needs to perform matrix computations. The user may not think that it is worth spending the time to write a program to calculate the result when there is an interactive interface that is ready to solve the problem immediately. For situations like the one just described, NetSolve offers three non-programming interfaces to choose from: MATLAB, Mathematica and Octave.

6.2.3 Load Balancing

An important aspect of many distributed and parallel applications is balancing the workload among the available processors as equally as possible. Since NetSolve is an application that splits multiple user requests among distributed resources, it must also be concerned about load balancing. In this case, load balancing means choosing the "best" computational server to carry out a particular request. Each agent stores information about every computational server in the system and uses that information to decide which machine to send the request to. There are many factors that influence the selection of the "best" server. For each user request, these factors are:

- Size in bytes of the user input data
- Size in bytes of the computed result
- Size of the problem
- Network latencies and bandwidths
- Algorithmic complexity
- Raw performance of the computational servers
- Workload of the computational servers

Given all these parameters, the NetSolve agent ranks the available computational servers based on their relative performance and returns this ranked list to the client.

6.2.4 Fault Tolerance

Occasionally, a NetSolve server will unexpectedly become unavailable. The most common causes for this unavailability include failure of the server hardware, problems with the network connection, and failure of the software. The NetSolve system should be able to identify and cope with any error that occurs without excessive performance degradation.

Failures in the NetSolve system are detected when a NetSolve process attempts to make a TCP connection to a server or they may be detected while a problem is being solved. The process then notifies the nearest agent, which takes the error into account. The failed server is marked as "bad," but it is only removed totally from the system if it has not restarted after a given amount of time has elapsed (typically 24 hours). In order to ensure that servers may be stopped and restarted safely, error reports must contain information to determine whether the server was restarted after the error. This information prevents a server from being marked "bad" because of an old failure report that was delayed in the network.

Once some failure has been detected in the NetSolve system and the server has been marked "bad", the agent ensures that no client will try to contact the failed server in the future. This is accomplished by sending the client an ordered list of servers which can fulfill the request. The client starts at the beginning of the list and tries servers in order until the problem is accepted and solved by a computational server or the list is exhausted and the problem still has not been solved. In the latter case, the client requests another ordered list of servers. This new list should be different from the previous list since each time a server from the first list failed, the client notified the agent of the failure and the agent marked the corresponding server as "bad." Thus, once the agent knows that a server has failed, it will exclude that server from future lists that it sends. This means that a client does not have to spend time trying to contact a server that the agent knows is not working or is unavailable.

6.2.5 Integration of PEBBLES into NetSolve Computational Servers

NetSolve was designed so that the administrator of a NetSolve system could add additional libraries to a server as easily as possible. This is accomplished by first creating a formal description of each function in the library—the IDL file (Interface Definition Language). The formal description contains the function name, the number of inputs and outputs and other properties of the library. The formal description is then translated into C source code by a compiler-like utility. Finally, the resulting C code is incorporated into the computational server (see Fig. 6.2).

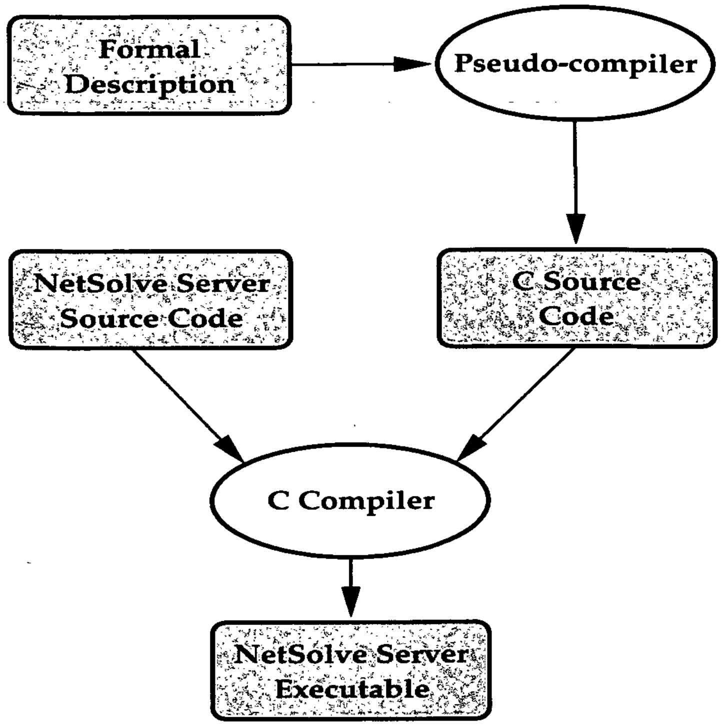


Figure 6.2: Integration of Software into a NetSolve Server.

The IDL file (formal description) for PEBBLES is given below. The first line contains the name of the library followed by the calling sequence. Each argument states if it is input or output, the type and name and for vectors / matrices the dimension (sparse matrices also contain the data structures for the compressed sparse row format in angular brackets). The calling sequence is followed by a description of the problem, the necessary libraries during compilation of the server and other options depending on the problem.

```
PROBLEM pebbles
C ROUTINE pebbles(IN double A[n][n]<nnz,index,pointer>,
                  IN double RHS[n],
                  INOUT double X[n],
                  IN string opt,
                  IN int n,
                  IN int nnz,
                  IN int index[nnz],
                  IN int pointer[n])

"iterative solver for A*x=b with A sparse and positive
definite (uses AMG as preconditioner)"
LIBS = "-L$(NSPEBBLES_LIB) -lpebbles -lstdc++"
NONMOVEABLE
```

So we need to provide a library `libpebbles.a` and also have to take special care that this library can be linked against NetSolve. This is because NetSolve is written in C and PEBBLES in C++. Therefore, PEBBLES is compiled as usual but we provide our own main file with an *extern "C"* directive (Stroustrup [52]). Of course, we also have to provide `libstdc++.a` when linking a C++ library against C. The keyword NONMOVEABLE states that the problem can not be moved between servers by NetSolve.

All necessary files to include PEBBLES into NetSolve and detailed instructions are contained in the subdirectory `netsolve_helper` of `fes_solver.tgz`.

6.3 HARNESS

HARNESS (Heterogeneous Adaptable Reconfigureable Networked SystemS) is an experimental metacomputing system (Beck et al. [9]) built around the services of a highly customizable and reconfigureable Distributed Virtual Machine (DVM).The virtual machine (VM) terminology, borrowed from PVM (Geist et al. [22]), refers to the fact that the computing resources on which a system runs can be viewed as a single large distributed memory computer.

6.3.1 Architecture

The virtual machine is a software abstraction of a distributed computing platform consisting of a set of cooperating daemon processes. In HARNESS applications obtain VM services by communicating with daemon processes through system-specific mechanisms encapsulated by a portable API. A DVM is defined there as a cooperating set of daemons that together supply the services required to run user programs as if they were on a distributed memory parallel computer. These daemons run on (often heterogeneous) distributed groups of computers connected by one or more networks.

Flexibility in service components comes from the fact that the HARNESS daemon supplies DVM services by allowing components which implement those services to be created and installed dynamically. Thus, the daemon process, while fixed, imposes only a minimum invocation structure on the DVM.

The HARNESS distributed registry service is used to hold all DVM state. When components are added to the DVM at invocation or runtime, this information is added to the registry. Similarly, the components of two DVMs can be merged en masse by merging their respective registries, and some set of components can be split from a DVM by creating a new registry for them and deleting their entries from the old one.

Fig. 6.3 gives a glimpse on the architecture of HARNESS. The central entity is the virtual machine, which is built up by a number of different machines and can be anything from a supercomputer to a PDA. On these computers runs the core which registers itself to a coherent replicated name service. The core is a daemon composed of a kernel and a set of required components (e.g., message passing, starting processes and threads, ...) which responds to requests from a local application or a remote daemon. The kernel together with a basic set of services constitutes a fully functional grid resource and is known as a HARNESS core (HCORE). This core can also be enhanced by user features through dynamically changeable components known as 'plug-ins'. These plug-ins can be either available from the local disc or through a remote repository. Currently, there are implementations for downloading components from a web repository and from IBP (Plank et al. [41]) and eXNode (Atchley et al. [5]).

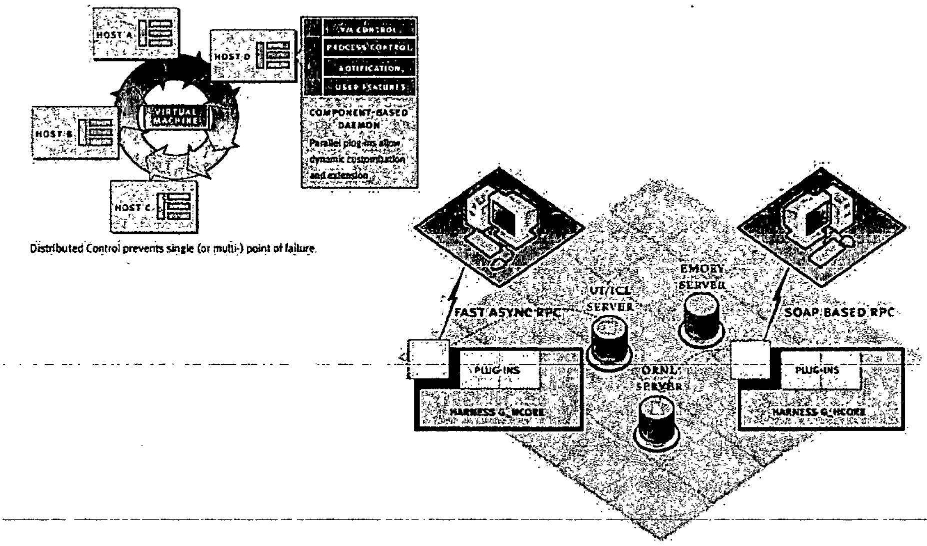


Figure 6.3: The HARNESS architecture: on the left the distributed nameservice which prevents a single point of failure and on the right the plug-in system.

When a client accesses Harness it includes the following steps: First it talks to the name service to look up and join a distributed virtual machine. Then it requests a package on one or more hcores, which is either already available or will be dynamically installed. If it has to be installed it is loaded from a repository and the name service is informed by the hcore that this package is now also available. The hcore then reports back to the client that it is ready and the client request the hcore to process the problem.

6.3.2 Including PEBBLES into HARNESS

To include PEBBLES into the FES-Tool using HARNESS consists of two steps. First, we have to build again a wrapper for pebbles to create a static library accessible from C. This library is used to build a plugin for HARNESS and is put on a repository / webserver. The second step consists of writing a MEX-function to access HARNESS from within MATLAB.

The whole procedure is more difficult than in NetSolve and also subject to change since HARNESS is still not released and currently only availabe as development version directly from Graham Fagg of the University of Tennessee in Knoxville. The advantage of HARNESS is its higher flexibility and faster execution compared to NetSolve and also for very big problems it shows more robustness – see Chapter 7 for more details.

The full implementation with instructions how to set up a DVM and access it from within MATLAB can be found in `fes_solver.tgz`, subdirectory `harness_helper`.

6.4 Interface to the FES-Tool (`solver.m`)

The previous sections in this chapter describe tools how to solve or remotely solve the system of linear equations arising in the FES-Tool. To tie together all the algorithms and implementations mentioned so far the MATLAB function `solver.m` was developed to provide all these functionalities behind an easy and flexible to use interface.

The FES-Tool provides the data for the system matrix A either in sparse matrix format or through the following vectors:

- `HD` the main diagonal of A,
- `ND1, ND2, ND3` the off-diagonals (above and below) of A, the length of the respective vectors determines their location,

The disturbance vector `b` gives the position of the electrodes and additionally a structure field `flag` is provided with which the solver is configured—see Appendix A for details. The result is returned as the sparse vector `x`. The complete calling sequence for the solver is therefore

```
x=solver(A, b, flag)
```

or

```
x=solver(HD, ND1, ND2, ND3, b, flag).
```

The function `solver.m` and other auxiliary routines are contained in the subdirectory `solver` of `fes_solver.tgz`. In the next chapter all the implementations described so far are evaluated and a recommendation for the use in the FES-Tool is given.

Chapter 7

Numerical Experiments

This chapter describes in detail the data tests and results to evaluate the discussed algorithms. Since there is a wide range of available algorithms and implementations and also various scenarios which data to use for the experiments a bottom-up approach is used. We start with a description of the available hardware and test data. In the section Parameter Analysis we study the influence of the various configuration options and afterwards all available solvers are compared by their run-time behavior. The chapter is concluded by the evaluation of tools for remote computing to leverage external computing resources.

7.1 Test Environment

At first the hardware used for testing the FES solver is described. The choice which hardware to use for the various tests depends on the following criterias:

- 32 or 64bit architecture—important for large scale problems,
- Windows or Unix operating system,
- availability of software—especially MATLAB but also services like HARNESS (Section 6.3),
- network connection—in terms of bandwidth to the Department of Biomedical Engineering and Physics and access of special ports because of security restrictions.

The central point, however, was the requirement to run the solver as part of the FES-Tool on the computer of the Department of Biomedical Engineering and Physics using MS Windows and the graphical user interface of MATLAB. The configuration of the workstation is given below.

Simulant2
AMD Athlon 2400+ 2GHz, 133Mhz Bus
3GB Ram, 128kB 1st level cache, 256kB 2nd level Cache
MS Windows 2000 Professional, MATLAB 6.5 R13
Cygwin 1.3.17, gcc 3.2

Since Simulant2 is a 32bit system and tests have shown that large matrices can not be solved due to memory restrictions, a 64bit machine at the Vienna University of Technology was also used.

Apollo: apollo.anum.tuwien.ac.at
Compaq SC45 with 4 Alpha-EV68 processors with 1GHz
16GB Ram, 8MB Cache/CPU
Compaq Tru64 UNIX V5.1A, MATLAB 6.5 R13, gcc 3.2

To actually use a 64bit computer it was necessary to utilize it remotely via NetSolve. Because of problems with installing NetSolve on Apollo, an HP Itanium located at the Innovative Computing Laboratory (ICL) at the Computer Science Department of the University of Tennessee in Knoxville was used.[1]

HP Itanium: hp04.cs.utk.edu
Dual Itanium Processor with 800MHz
2GB Ram, 2048KB Cache
Red Hat Linux 7.3, Kernel 2.4.18, gcc 2.96

For the sake of completeness the information about the machines where HARNESS was tested is listed here, too. Since HARNESS is still in its early stages of development - although already fully functional - it is currently not available for Windows. Hence, it was tested only on the following machines at ICL.

Microsoft Cluster: msc*.cs.utk.edu
Dual Pentium III with 933MHz
512MB Ram, 256kB Cache
Red Hat Linux 7.3, gcc 2.96

Cetus Cluster: cetus*.cs.utk.edu
Sun Sparc Ultra1 with 502MHz
512MB Ram
Solaris 8, gcc 2.95.3

Beside of the hardware the test data used to evaluate the algorithms and implementations is described next. In the previous chapters the Poisson problem (4.2) was considered. This is the simplest non-trivial partial differential equation in 3 dimensions and is from now on referred to as "Model I."

On the other hand, there are two data sets of computer tomography (CT) images (the left and right thigh of a paraplegic patient) that are currently only available in the full resolution of 314×266×350 and a scaled down set with a resolution of 314×266×51. Because the full data would lead to matrices that could not be handled on todays computers only the reduced data set is used and it is available with different electrode configurations. The various electrode configurations of

---

[1] With the availability of the final version of NetSolve 2.0 the problems with Apollo have been solved. But since it is necessary to register the IP addresses which are accessed from Simulant2 at the Vienna General IT-dept. and the trouble-free usage of the machines at the ICL the HP Itanium was chosen as 64bit server.

this model are given as number pairs specifying the position of the anode and cathode at the full resolution. E.g., 100/250 is the shortcut for: The middle of the anode is at the 100^{th} cross-section and the middle of the cathode at the 250^{th} cross-section.

Since Model I is quite different to the model based on the CT data we created another model problem because the resulting systems are also quite large and currently only available in 4 different configurations. The requirements for this model are that it should feature similar properties as the CT data (inhomogeneity, anisotropy), be highly configurable (in terms of size, conductivity values and tissue configuration) and easy to build up.

Therefore, we built a 3-dimensional conductivity matrix that mimics the tissue structure of a thigh, i. e., it contains muscles, fat and a bone. Skin and blood have been left out because the model problem is aimed for smaller problem sizes and therefore at this discretization level it would not make sense to simulate the behavior of skin or blood vessels. This matrix can be built up in arbitrary sizes in any direction and Fig. 7.1 shows the metrics for a 16 × 16 × 16 model problem—other sizes are calculated relative to the given numbers. Table 7.1 lists the used conductivity values of this model which will be named "Model II." For consistency reasons we refer to the set of CT data as "Model III."

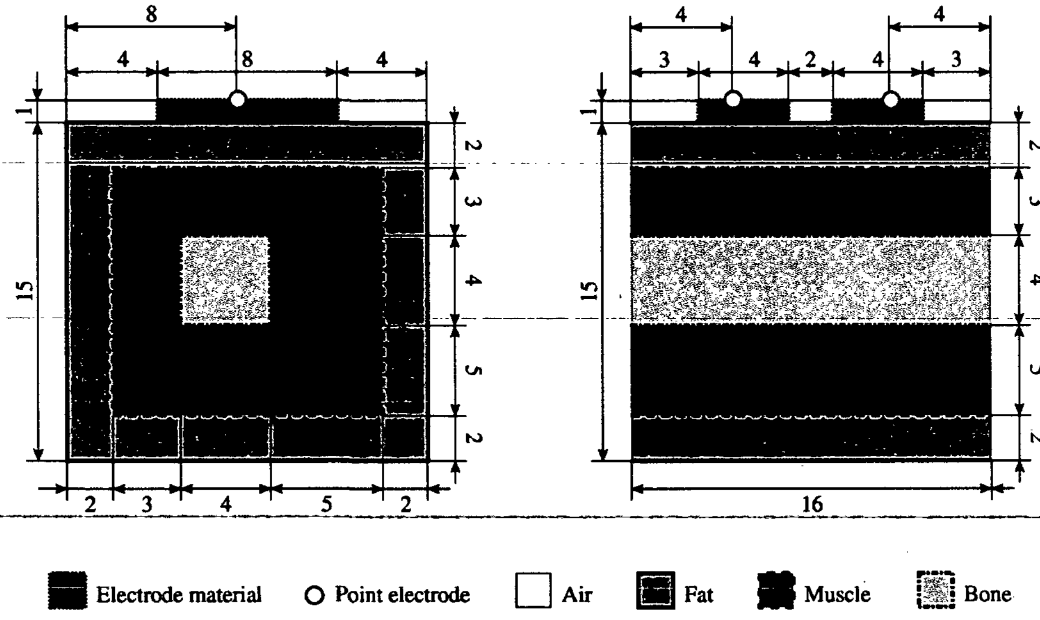


Figure 7.1: Metrics of Model II: on the left a cross section and on the right a view from the side. The given values are the scaling factors to a 16 × 16 × 16 model.

Based on the conductivity matrix that represents the specific conductivity in each voxel the absolute conductivity is calculated as described in Chapter 2. When

Conductivity	Value
electrode material	10
muscle transversal	0.1
muscle longitudinal	0.7
fat	0.03
bone	0.016
air	0

Table 7.1: Conductivity values in S/m of the various materials and tissue types in Model II (*corresponds to the values of the real data*).

using Model II the following limitations have to be kept in mind.

- The simplified geometry does not take into account the cylindric form of the thigh and especially it is completely homogeneous in longitudinal direction.
- On generation of Model II the damping factor of the Neumann boundary conditions at both ends is neglected. Therefore, the behavior at the first and last cross-sections will be inaccurate.

For the following discussion of Model II we used the standard configuration (anisotropy in longitudinal direction, default conductivity values), a model size of 32 × 32 × 32 and solved the system with the MATLAB function `pcg` (tolerance for the relative residual is 10^{-6}).

We will present 2 dimensional contour plots of cross-sections. The colors of this equipotential lines represent values from -1 (blue) to 1 (red) and lines are at 0.1 V steps in the thigh and at 0.004 V steps in the electrodes. This allows the visualization of the point electrodes within the electrode material. The axis description is "Pixel" in the respective directions but is omitted due to clearer representation.

The other type of figures show line plots of a virtual muscle fiber in longitudinal direction and the electric potential distribution within this single fiber. The x-axis is the longitudinal direction given in pixels, while the y-axis the results in Volt or respectively Volt/pixel (1st derivative), Volt/pixel^2 (2nd derivative) represents.

In the next two Figures 7.2 and 7.3 the voltage distribution in Model II is shown using contour plots in MATLAB. While the first figure shows an evenly distributed symmetrical electric field indicated by the potential lines, the second figure is slightly asymmetric because of the bone which is not in the middle but shifted to the upper left of the artificial thigh and additionally the point electrodes are not exactly in the center of the model—Fig. 7.3 shows the 10th cross-section with the anode and and the 24th cross-section with the cathode. Because of the homogeneity in longitudinal direction the left and right field distribution are identical except for the sign of the voltage values which are represented through the different colors.

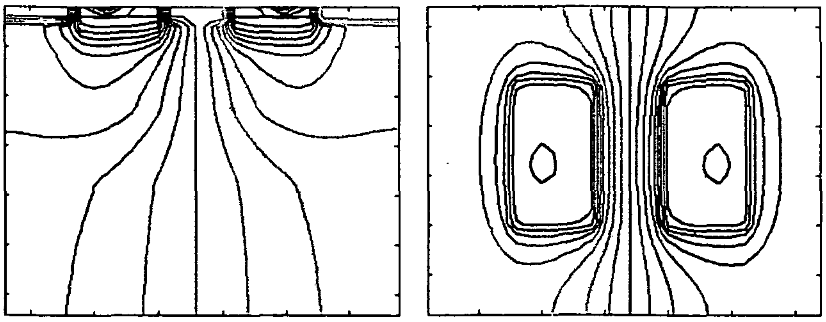


Figure 7.2: Contour plots of the voltage distribution indicated by equipotential lines—spacing: 0.1 (thigh), 0.004 (electrode material)—of the middle (16^{th}) longitudinal cross-section and of the 3^{rd} layer of the model problem.

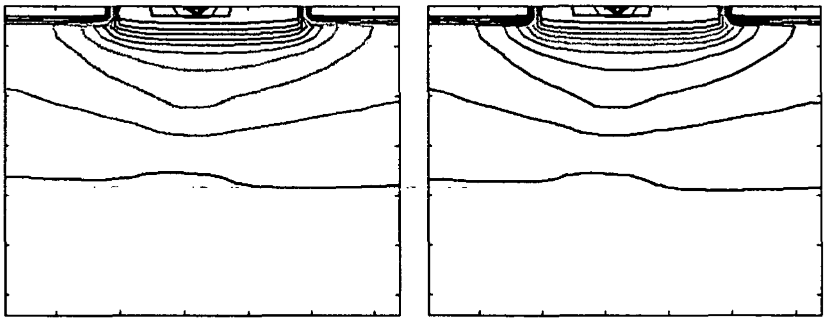


Figure 7.3: Contour plots of the voltage distribution in cross-sections below the anode (left) and cathode (right) indicated by equipotential lines—spacing: 0.1 (thigh), 0.004 (electrode material).

Another important property is the voltage distribution in longitudinal direction (the main direction of the muscle fibers) and its derivatives. These characteristics together with the current/voltage and frequency of the stimulation give information about the activation of the muscle—see Reichel [44] for a detailed description.

Fig. 7.4 gives an overview of the solution and its derivatives in different layers below the point electrodes. Various informations can be obtained from this array of contour plots. In each row the curves get smoother the deeper the layer. This cleary demonstrates the difficulty to stimulate muscles with surface electrodes because the stimulation has to overcome the barrier of fat. In each column we see that a single peak in one graphic is represented with two peaks in the graphic below which again verifies the results.

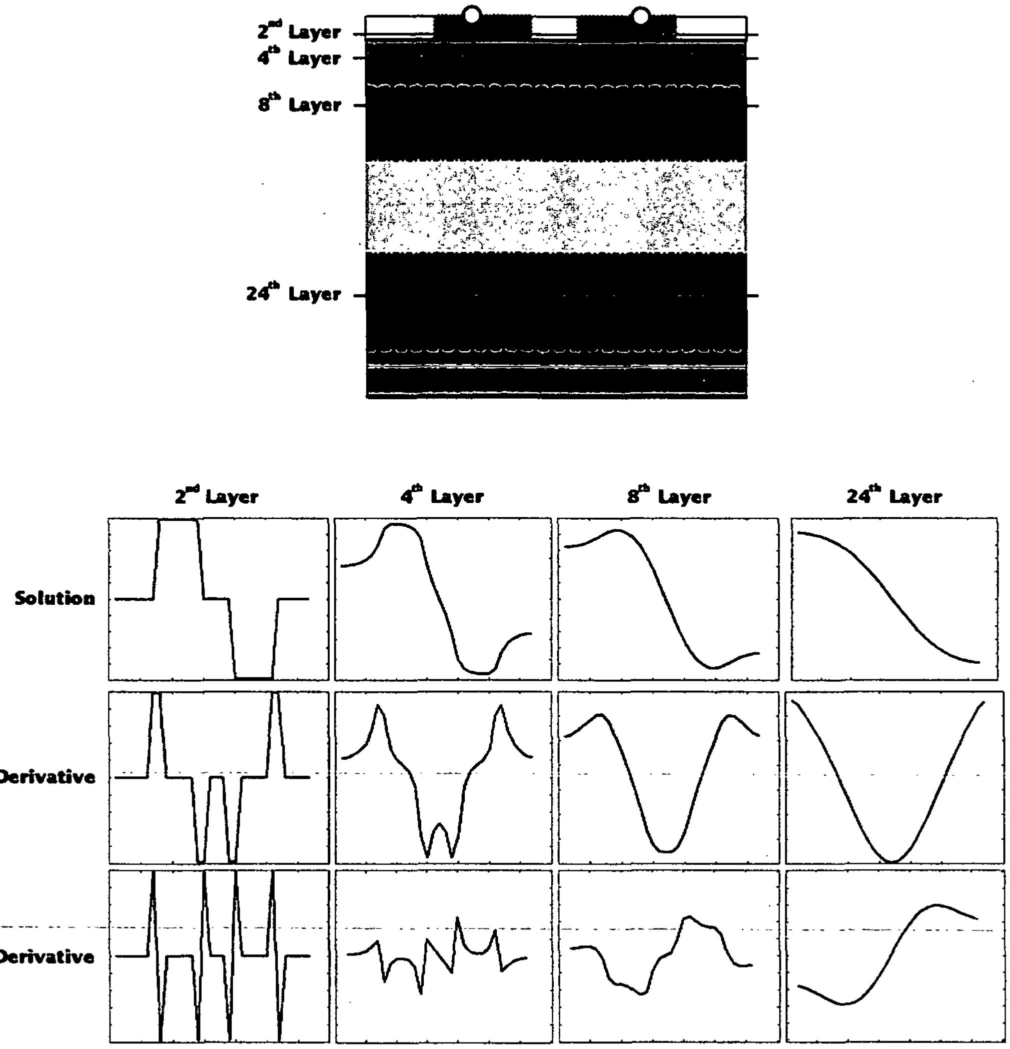


Figure 7.4: Overview of the characteristics of the solution and its derivatives. At the top is a longitudinal cross-section of Model II with lines at 4 different layers. Below line plots of a "virtual" muscle fiber at this layers show the electric potential distribution and the 1st and 2nd derivative.

Figures 7.5 and 7.6 show the voltage in longitudinal direction below the point electrodes in different layers. While the different electric potentials in the first figure are still clearly seen the top of the muscle (8th layer) already shows a much smoother curve and espically note the smaller range in the y-axis.

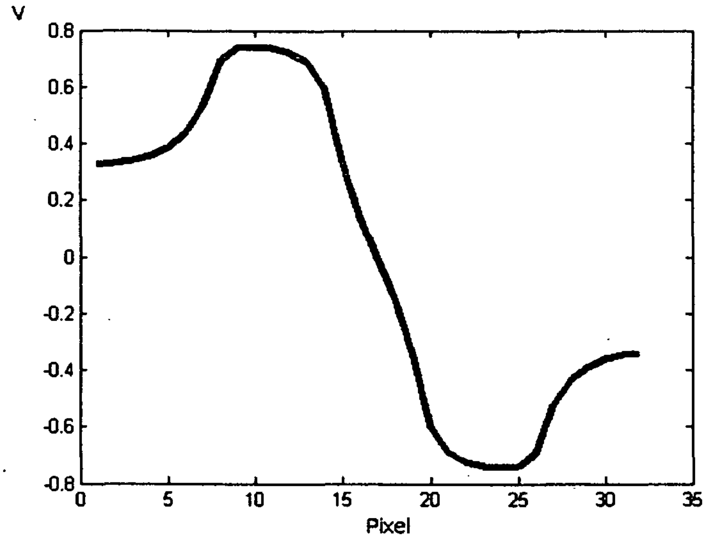


Figure 7.5: Voltage in longitudinal direction in the 4th layer (fat).

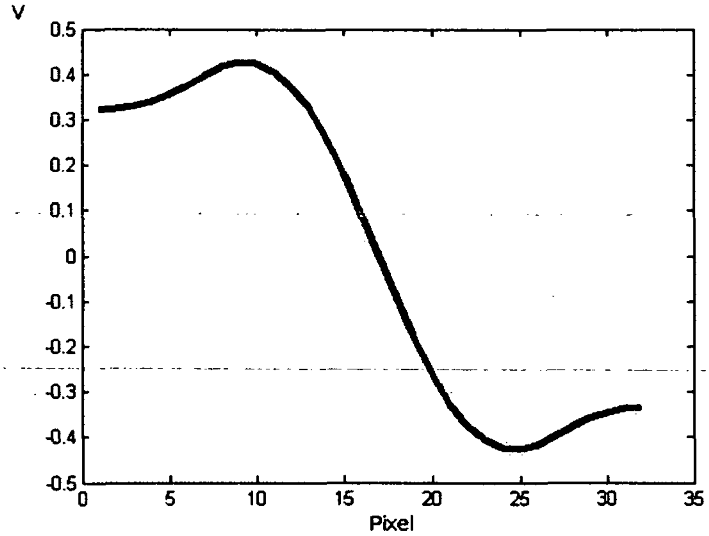


Figure 7.6: Voltage in longitudinal direction in the 8th layer (muscle).

Figures 7.7 and 7.8 show the characteristics of the first and second derivatives of the voltage distribution in the longitudinal direction below the point electrodes. These results are especially important since the extreme values of the first derivative are areas of activation at muscle tendon junctions (hemidesmosomes) and the second derivatives describe activations over the entire length of muscle cells.

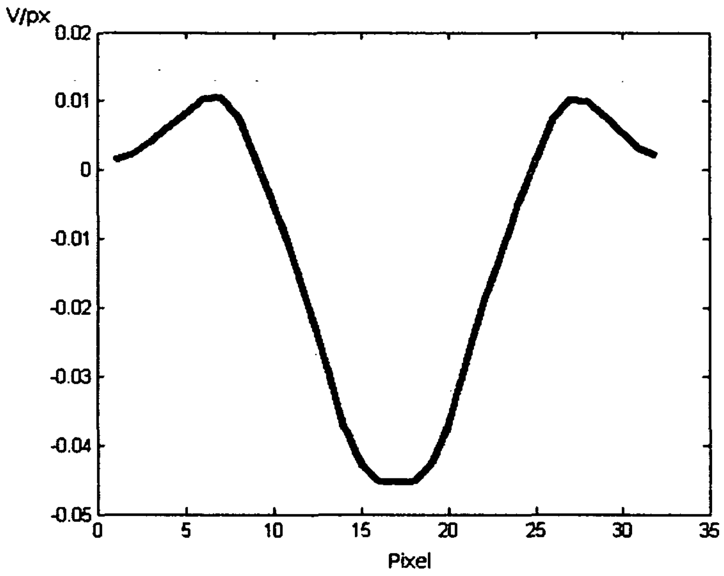


Figure 7.7: Characteristics of the first derivative of the voltage in longitudinal direction.

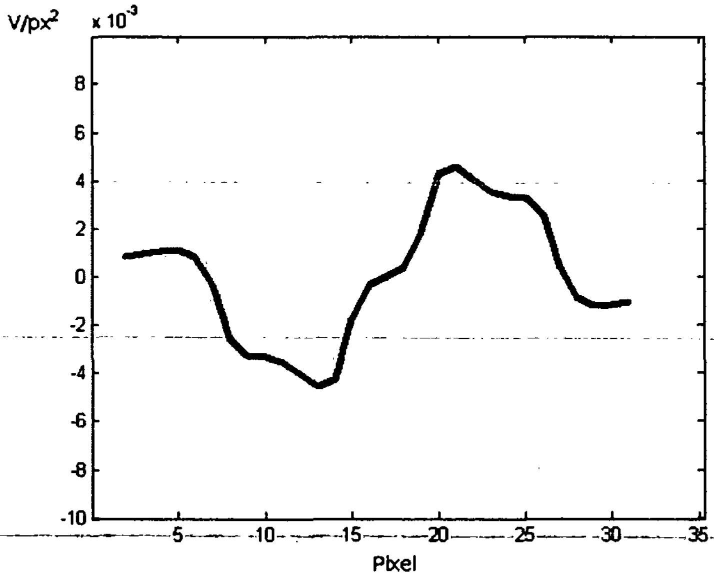


Figure 7.8: Characteristics of the second derivative of the voltage in longitudinal direction.

In the following section the numerical properties of the available data sets are compared. The aim is to show conformance and shortcomings of the Model I and Model II compared to Model III and what conclusions could be drawn from the numerical experiments.

7.2 Data Set Analysis

As stated above an important measure for any given linear system is its condition number (3.7) where large numbers indicate high sensitivity of the solution to errors in the data. Table 7.2 shows the condition number for various sizes of Model I, Model II and down scaled CT data[2]. While the results of Model II are of the same order as Model III Model I already shows differences. These differences are due to smooth coefficients of the Poisson model problem and the inhomogeneity within the other data sets.

Model I		Model II		CT data	
size	κ	size	κ	size	κ
$16 \times 16 \times 16$	$1.717 \cdot 10^2$	$16 \times 16 \times 16$	$3.959 \cdot 10^5$	$105 \times 89 \times 3$	$6.927 \cdot 10^5$
$24 \times 24 \times 24$	$3.875 \cdot 10^2$	$24 \times 24 \times 24$	$1.155 \cdot 10^6$	$157 \times 133 \times 3$	$2.245 \cdot 10^6$
$32 \times 32 \times 32$	$6.897 \cdot 10^2$	$32 \times 32 \times 16$	$1.471 \cdot 10^6$	$105 \times 89 \times 11$	$3.980 \cdot 10^6$
		$32 \times 32 \times 32$	$2.676 \cdot 10^6$		

Table 7.2: Condition of the various data sets in different sizes.

In the previous chapters the various implementations were compared by their conververgence speed, i. e., the number of iterations it takes until the relative residual is below a given boundary. Here we will again use convergence speed to compare the three models by using PCG in MATLAB without preconditioning. Model I in Fig. 7.9 shows a steady drop of the relative residual and although no preconditioner is used the solver converges within a reasonable number of iterations. Model II (Fig. 7.10) gives good results for small sizes but larger matrices start to need a long run-time until the relative residual starts to drop slowly as in Model III (Fig. 7.11). Again Model II shows a similar behavior as Model III while Model I differs. Also note the very similar convergence behavior of the different electrode configurations in Model III. Therefore, in the following we will always use the configuration 100/250 when comparing Model III with other model problems.

Another important point when working with such data sets is the accuracy with which all calculations are performed. Because all input data already include some error it would not make sense to use valuable computational power to go for unreasonable "high accurate" results. For the given problem the FES-Tool is used to simulate the activation of muscles and, as noted above, the first and second derivatives of the solution from the PDE are used for this purpose. Therefore, we are only interested in the results at the areas of the muscle tissue and in the

---

[2] Since the complete systems of the CT data (Model III) are way too large to compute the condition number some scaled down CT data sets from previous experiments were used.

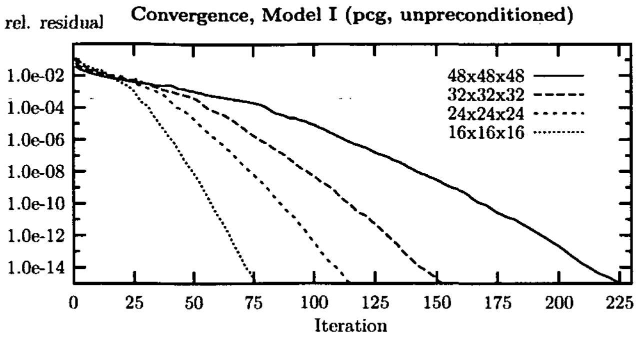


Figure 7.9: Convergence behavior for Model I.

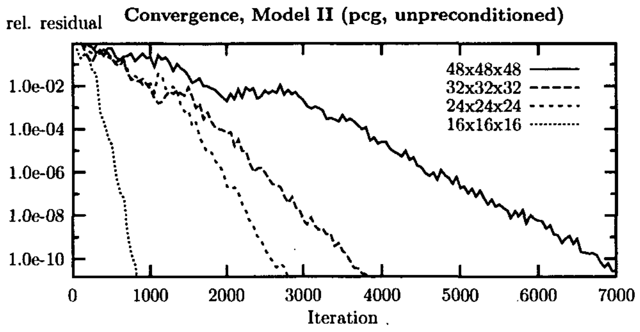


Figure 7.10: Convergence behavior for Model II.

following all data is restricted to this area. This is done by applying a mask to the results and eliminating all of the voxels but those of tissue type muscle. Since Model I has shown that it does not match the properties of Model III so far it is left out in further tests.

Table 7.3 shows the results of solving the problems at various accuracies and calculating the second derivative in longitudinal direction. The error ε of the solution x is given respective to the peak-to-peak-value of the highly accurate reference solution x_{ref} obtained with relative residual 10^{-12} in the region of interest (muscle) and is calculated by

$$\varepsilon = \frac{\max(x_{ref} - x)}{\max(x_{ref}) - \min(x_{ref})}. \tag{7.1}$$

As the results suggest a relative residual of 10^{-2} is too crude. Starting with an accuracy of 10^{-4} we already get good results and from now on we will use 10^{-6} as tolerance for iterative solvers which assures a good quality of the solution.

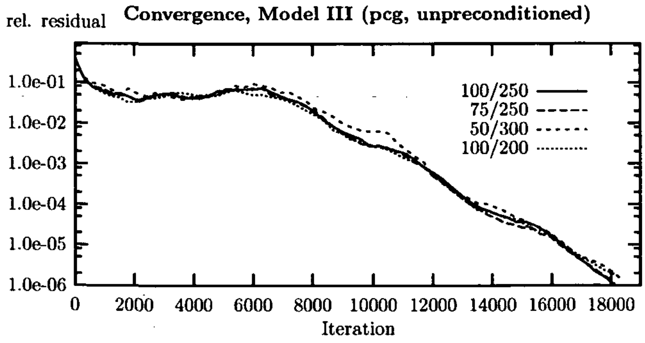


Figure 7.11: Convergence behavior for Model III. All 4 system are based on the same thigh with different electrode configurations as described before (the number pair indicates the position of the anode/cathode.

size	Model II rel. residual	ε	configuration	Model III rel. residual	ε
$16 \times 16 \times 16$	10^{-2}	$3.887 \cdot 10^{-3}$	100/250	10^{-2}	$1.749 \cdot 10^{-1}$
	10^{-4}	$1.409 \cdot 10^{-5}$		10^{-4}	$6.476 \cdot 10^{-4}$
	10^{-6}	$1.455 \cdot 10^{-6}$		10^{-6}	$8.957 \cdot 10^{-6}$
	10^{-8}	$9.925 \cdot 10^{-9}$		10^{-8}	$2.887 \cdot 10^{-8}$
	10^{-10}	$4.111 \cdot 10^{-11}$		10^{-10}	$2.525 \cdot 10^{-10}$
$32 \times 32 \times 32$	10^{-2}	$2.089 \cdot 10^{-2}$	50/300	10^{-2}	$2.999 \cdot 10^{-1}$
	10^{-4}	$2.016 \cdot 10^{-4}$		10^{-4}	$2.582 \cdot 10^{-4}$
	10^{-6}	$2.857 \cdot 10^{-6}$		10^{-6}	$3.786 \cdot 10^{-6}$
	10^{-8}	$2.476 \cdot 10^{-8}$		10^{-8}	$3.842 \cdot 10^{-8}$
	10^{-10}	$8.325 \cdot 10^{-11}$		10^{-10}	$2.762 \cdot 10^{-10}$

Table 7.3: Error in the second derivative of the solution in the muscle when using different tolerances.

7.3 Parameter Analysis

In this section we study the impact of the various parameters on the quality of the solution and the run-time performance. The quality of the solution is measured by the error respective to the peak-to-peak-value in the region of interest as described in (7.1). This error is calculated for the solution (ε_0), the first (ε_1) and the second derivative (ε_2). In addition to the maximum we also provide the respective mean values in the tables in a second row (e. g., Table 7.4). All tests were performed with the following base configuration if not otherwise specified: `type`=0, `modifier`=1, `tol`=10^{-6}, `solver`=*pcg*, `precond`=*amg*, `service`=*local*.

We start with the analysis of the resolution. Since the available set of data is too large (314 × 266 × 350) to be processed it is currently scaled down to a resolution of 314 × 266 × 51. We are now interested in the sensitivity of the solution if it is solved on a lower resolution and the result interpolated and compared to the solution of the original size. These tests are performed only with Model II because the FES-Tool currently does not support any other resolutions then the one mentioned above. As reference solution we take a system of the size 32 × 32 × 32 and compare it to systems scaled down equally in each spatial dimension and scaling it down in only one dimension. The interpolation method used in MATLAB is 'spline'. Table 7.4 shows the results of this test.

size	ε_0	ε_1	ε_2
16 × 16 × 16	$1.036 \cdot 10^{-1}$	$3.628 \cdot 10^{-1}$	$6.084 \cdot 10^{-1}$
	$1.453 \cdot 10^{-2}$	$5.046 \cdot 10^{-3}$	$2.001 \cdot 10^{-3}$
24 × 24 × 24	$1.032 \cdot 10^{-1}$	$1.462 \cdot 10^{-1}$	$4.846 \cdot 10^{-1}$
	$2.686 \cdot 10^{-2}$	$1.644 \cdot 10^{-3}$	$8.191 \cdot 10^{-4}$
32 × 32 × 16	$6.991 \cdot 10^{-2}$	$3.877 \cdot 10^{-1}$	$5.039 \cdot 10^{-1}$
	$1.698 \cdot 10^{-2}$	$4.965 \cdot 10^{-3}$	$1.947 \cdot 10^{-3}$

Table 7.4: Error of various resolutions if interpolated and compared to the model problem of size 32 × 32 × 32.

The results in Table 7.4 are not satisfying and require a further investigation. First we note that the maximum error is quite high, but the mean error is at a much more moderate level. The contour plots of the eletric potential is shown in Fig. 7.12. Note the contour lines at the interpolated solution which are wider apart at the end of the electrodes which is also the place where the largest errors occur. Within a cross-section the largest error is at the crossover from fat to muscle—see Fig. 7.13. Finally, we compare the concrete values of the second derivative in longitudinal direction where the largest errors occur with the reference solution. Fig. 7.14 shows the qualitatively similar characteristics of both results but some discrepancy at various sampling points.

Another possible source of the error for the system with resolution of 24 × 24 × 24 is the fact that 24 is not a multiple of 16 and therefore the generated conductivity matrix is not an exactly scaled version of the original because of roundoff errors which leads to a slightly different system.

Regarding the resolution of 32 × 32 × 16 one should note—although four times bigger than the smallest system—the results are not that much better. This suggests that an evenly scaled down system would give better results (when taking run-time performance into account) than a minimization in only one direction. Since the test is performed only on Model II it is recommendable to repeat it with Model III and verify these results.

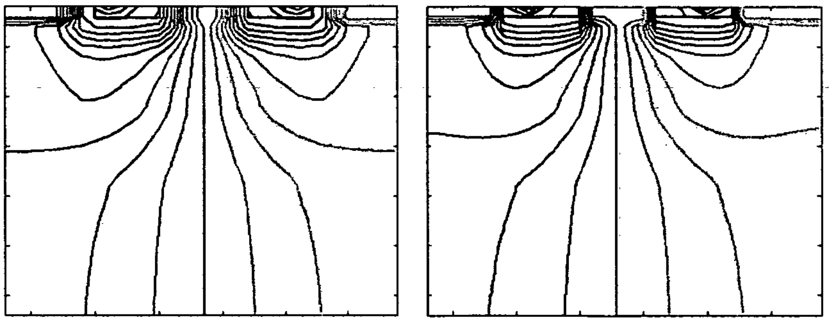


Figure 7.12: Comparison of contour plots of the interpolated results (left) and the solution with the full data set (right).

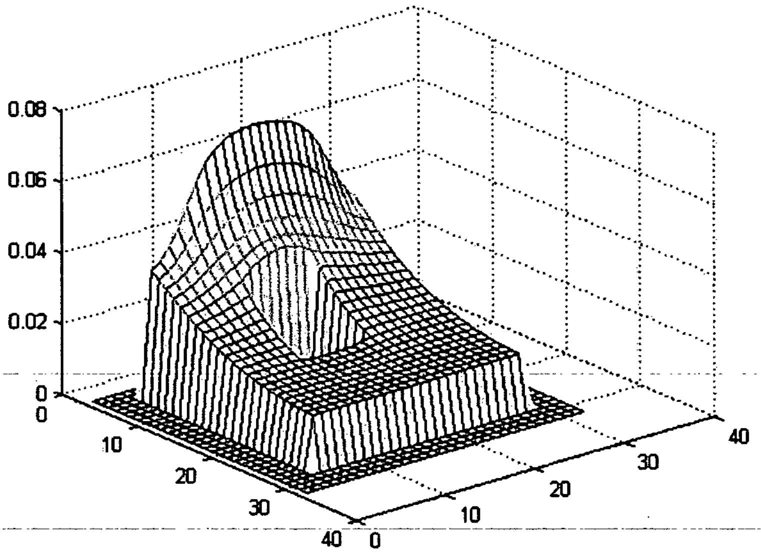


Figure 7.13: Mesh-plot of cross-section with the largest error in the electric potential field. The largest error is at the crossover from fat to muscle below the electrode (which is hidden behind the peak).

When the FES-Tool was developed at the Department of Biomedical Engineering and Physics an important aspect was that the anisotropy of the muscle should be taken into account. Table 7.5 shows the results comparing data sets with and without anisotropy with the reference solution taking the anisotropy into account. Additionally, the run-time performance was evaluated and for large systems a speed-up of more than 10 % was measured.

As the results show, is the effect of the anisotropy more clearly seen for Model

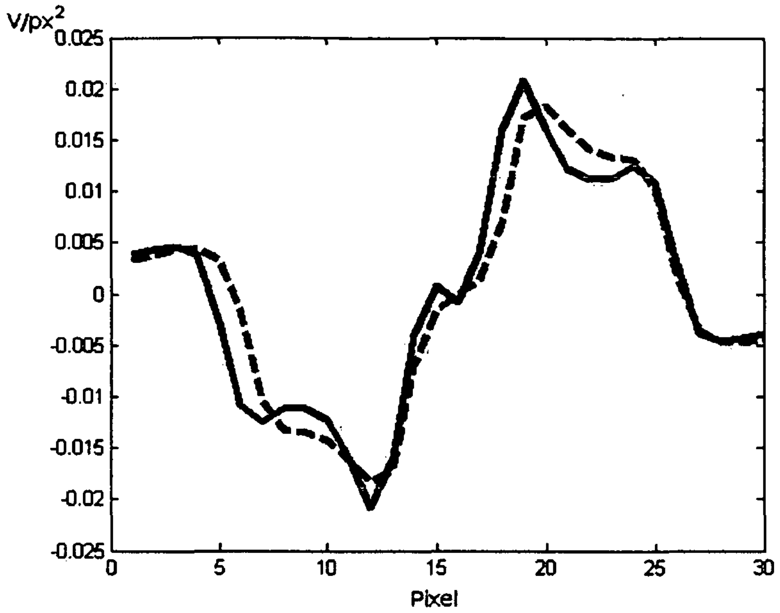


Figure 7.14: Comparison of the second derivative of the electric potential along the muscle fiber with the largest error. The interpolated solution is shown as solid red line against the reference solution with a dashed blue line.

	ε_0	ε_1	ε_2	speed
Model II – 32 × 32 × 32	$9.841 \cdot 10^{-3}$ $4.194 \cdot 10^{-3}$	$1.789 \cdot 10^{-2}$ $2.675 \cdot 10^{-3}$	$1.808 \cdot 10^{-2}$ $1.127 \cdot 10^{-3}$	100 %
Model III – 100 / 200	$5.892 \cdot 10^{-2}$ $3.101 \cdot 10^{-2}$	$1.012 \cdot 10^{-1}$ $5.180 \cdot 10^{-3}$	$8.823 \cdot 10^{-2}$ $2.967 \cdot 10^{-3}$	88 %

Table 7.5: Results of comparing data without anisotropy to the reference solution with anisotropy.

III. This could be, on the one hand, a shortcoming of Model II or, on the other hand because of the relative small size. Nevertheless, this experiment shows the importance of the anisotropy and its influence on the quality of the solution in Model III.

Model II and III use a pair of point electrodes and some artificial electrode material on top of the thigh to simulate the use of surface electrodes. The electrode material is defined as a tissue type with high conductivity but preliminary tests have shown that the conductivity values used have a decisive impact on the runtime behavior, i. e., the higher the conductivity value the longer the calculation.

With Model II and size 32 × 32 × 32 Table 7.6 compares various conductivity values g of the electrode material to the reference solution ($g = 10\ S/m$).

	ε_0	ε_1	ε_2	speed
$g = 1\ S/m$	$8.167 \cdot 10^{-2}$	$1.236 \cdot 10^{-1}$	$8.231 \cdot 10^{-2}$	53 %
	$3.609 \cdot 10^{-1}$	$2.153 \cdot 10^{-2}$	$4.752 \cdot 10^{-3}$	
$g = 5\ S/m$	$1.083 \cdot 10^{-2}$	$1.639 \cdot 10^{-2}$	$1.085 \cdot 10^{-2}$	89 %
	$4.762 \cdot 10^{-3}$	$2.843 \cdot 10^{-3}$	$6.267 \cdot 10^{-4}$	
$g = 50\ S/m$	$9.040 \cdot 10^{-3}$	$1.369 \cdot 10^{-2}$	$9.158 \cdot 10^{-3}$	133 %
	$3.971 \cdot 10^{-3}$	$2.365 \cdot 10^{-3}$	$5.239 \cdot 10^{-4}$	
$g = 100\ S/m$	$1.026 \cdot 10^{-2}$	$1.537 \cdot 10^{-2}$	$1.097 \cdot 10^{-2}$	141 %
	$4.470 \cdot 10^{-3}$	$2.656 \cdot 10^{-3}$	$5.885 \cdot 10^{-4}$	

Table 7.6: Results of comparing data with various conductivity values for the electrode material to the reference solution of $g = 10\ S/m$.

The results confirm prior tests of the run-time behavior and clear show that low values of g would lead to notably large errors.

Finally, we tested the impact of different starting values when performing iterative solution methods. Since the FES-Tool is used to test different electrode positions it would make sense to use results of one configuration as starting value for another—slightly different—configuration. Unfortunately, it turned out that this method does not result in any better run-time performance. The tests were performed with unpreconditioned PCG from MATLAB as well as PEBBLES but neither the number of iterations nor the run-time decreased.

7.4 Numerical Performance

In this section we analyze the run-time behavior of the various solvers described in the previous chapter. The tests are again performed with three data sets of Model II: 32 × 32 × 32 (dim=32 768), 48 × 48 × 48 (dim=110 592) and 64 × 64 × 64 (dim=262 144); as well as Model III (dim=1 697 265) with electrode configuration 100/250.

Four types of solvers are investigated:

- direct solvers and simple iterative solvers,
- iterative solvers provided by MATLAB,
- iterative solvers combined with preconditioners,

- multigrid solvers.

To test direct solvers (SuperLU and MA28) we leverage the power of the software repository of NetSolve. To avoid tampering of the results through transmission times these tests were performed locally in Knoxville at the ICL. Those solvers deliver a bad performance for medium sized systems and completely fail due to memory requirements for large systems. Table 7.7 shows the available results for direct solvers (systems larger than 32 × 32 × 32 could not be solved) and the result of Gauss-Seidel. Since Gauss-Seidel takes about 5 minutes per single iteration and Table 4.1 already showed the convergence behavior (more than 1000 iterations to achieve a relative residual of 10^{-6}) the exact value is not calculated.

	SuperLU	MA28	Gauss-Seidel
32 × 32 × 32	2 923 sec	7 792 sec	5 min/iter

Table 7.7: Runtime measurement of direct solvers and the Gauss-Seidel iterative solver for Model II.

Next we compare the iterative solvers provided by MATLAB in Table 7.8. It is clearly seen that the more robust solvers like `bicg` and `bicgstab` have noticeable longer solution times than `pcg`. Not included in this list is `gmres` because the implementation in MATLAB seems to make huge memory requirements and was only able to solve systems up to a size of 16 × 16 × 16 and fails with an *OUT OF MEMORY* error for larger systems. Especially interesting are the extremely good results of `minres` which is always faster than `pcg`.

	pcg	bicg	bicgstab	minres
Model II – 32 × 32 × 32	105 sec	140 sec	211 sec	65 sec
Model II – 48 × 48 × 48	690 sec	910 sec	1446 sec	407 sec
Model II – 64 × 64 × 64	2 272 sec	2 996 sec	4 304 sec	1 419 sec
Model III – 100 / 250	53 084 sec	74 055 sec	29 550 sec	32 434 sec

Table 7.8: Runtime measurement of iterative solvers from MATLAB without preconditioning.

Next we test these iterative methods with preconditioning as described in Section 5.2. Since it would be inefficient to use the incomplete Cholesky factorization for large sparse banded matrices with which we deal here, we restricted the tests to the block-tridiagonal Cholesky factorization. But already this optimized variant failed for systems larger than dim=32 768 (32 × 32 × 32) because of an *OUT OF*

MEMORY error in MATLAB. This and the extreme long time for calculating the preconditioning matrix render the use of this method as unsuitable. Test results with Model II ($32 \times 32 \times 32$) leads to a solution time of 16 sec when using pcg and 1 554 sec for calculating the preconditioning matrix.

Finally, we turn to multigrid methods. The results of the multigrid solvers in Section 5.3 were quite promising but unfortunately it turns out that a straight forward implementation of those basic principles cannot handle Model II and III. A standard Gauss-Seidel smoother with a simple coarse grid correction scheme will fail for the discontinuities and anisotropies of those problems. Therefore, we were only able to test with PEBBLES in the configuration of a preconditioned conjugate gradient method using AMG as preconditioner.

Table 7.9 shows the results of these tests and the distinct phases of of the algorithm. Because of memory requirements PEBBLES cannot solve larger systems and especially Model III. In the next section we will use tools for remote computing to access 64bit computers with which it will be possible to overcome these limitations—Table 7.10 gives the results of executing PEBBLES on such a 64bit computer (Apollo).

	complete	setup	solver	iterations
Model II – $32 \times 32 \times 32$	2.6 sec	1.0 sec	0.7 sec	8
Model II – $48 \times 48 \times 48$	9.5 sec	4.0 sec	3.2 sec	11
Model II – $64 \times 64 \times 48$		memory!		
Model III – 100 / 250		memory!		

Table 7.9: Results of using PEBBLES (pcg with AMG preconditioning) with various data sets on Simulant2.

Table 7.10 also includes the results of applying AMG (PEBBLES) to a standard problem (Model I) which could be easily be handled by geometric multigrid. It clearly shows that the additional cost in terms of the setup phase for the AMG algorithm is quite high.

7.5 Remote Computing

In Sections 6.2 and 6.3 we have described tools for remote computing, namely NetSolve 2.0 and HARNESS (still in development). The main reason to use this software here is to access 64bit computers which allow a larger address space and are able to handle the large matrices arising in the FES-Tool. Table 7.11 shows the run-time measurements of using a NetSolve 2.0 Client at the Department

	complete	setup	solver	iterations
Model I – 32 × 32 × 32	2.7 sec	1.8 sec	0.5 sec	6
Model I – 48 × 48 × 48	10.7 sec	7.3 sec	2.6 sec	7
Model I – 64 × 64 × 48	27.3 sec	18.2 sec	7.4 sec	8
Model II – 32 × 32 × 32	3.1 sec	2.2 sec	0.6 sec	8
Model II – 48 × 48 × 48	15.5 sec	11.3 sec	3.7 sec	11
Model II – 64 × 64 × 48	69.9 sec	32.3 sec	14.8 sec	17
Model III – 100/250	1 069.2 sec	823.4 sec	223.2 sec	28

Table 7.10: Results of using PEBBLES (pcg with AMG preconditioning) with various data sets on Apollo.

of Biomedical Engineering and Physics accessing the HP Itanium at the ICL in Knoxville, Tennessee.

	complete	solver	transfer-time
Model II – 32 × 32 × 32	56 sec	4 sec	52 sec
Model II – 48 × 48 × 48	181 sec	17 sec	164 sec
Model II – 64 × 64 × 48	489 sec	75 sec	414 sec
Model III – 100 / 250	3 679 sec	1 218 sec	2 461 sec

Table 7.11: Results of using PEBBLES (pcg with AMG preconditioning) on the HP Itanium via NetSolve from the Vienna General.

The big advantage of using NetSolve compared to Table 7.9 is that here all systems can be solved, but the cost of transfer is quite high. Since the transfer-time depends highly on the available bandwidth to the U.S. these numbers are of course subject to fluctuations. Nevertheless, the results are still available much faster than using iterative solvers from MATLAB as the results in Table 7.8 show.

Finally, we also give a speed comparison of NetSolve and HARNESS. Because there is currently no Windows Client available for HARNESS we tested on May 2003 NetSolve and HARNESS on a heterogeneous Linux / Solaris cluster at the ICL. Table 7.12 gives the mean values of data sets performed with both systems executing PEBBLES remotely. They show almost the same performance.

dim	NetSolve	HARNESS
28 035	2.48 sec	2.27 sec
102 795	8.23 sec	8.09 sec
280 350	34.00 sec	34.04 sec
417 620	53.16 sec	52.71 sec
626 430	96.72 sec	96.52 sec
918 764	119.62 sec	126.33 sec

Table 7.12: Comparison of using PEBBLES (pcg with AMG preconditioning) remotely with NetSolve and Harness.

Chapter 8

Conclusions

In this chapter we summarize our results and give recommendations how to optimally use the presented models and algorithms. Basically, we can draw conclusions in two areas. First we have tested and implemented all major solution strategies which are currently available for the problem arising in the FES-Tool at the Department of Biomedical Engineering and Physics. On the other hand we have developed a model that allows to study the behavior of the data based on CT images on a smaller scale which led to some new insight into this class of problems.

The studied solution methods can be categorized in four groups: direct methods, basic iterative methods, Krylov subspace methods and multigrid methods. As expected, direct methods and basic iterative methods are not suited to deal with the given class of problems, although we used specialized direct solvers and a Gauss-Seidel implementation that took advantage of the sparse symmetric structure of the matrices.

Valuable insight was gained when Krylov subspace methods were used. On the one hand, with these algorithms we were able to solve even the largest systems on the local 32bit computer and, on the other hand, they showed the importance of good preconditioners for the poorly conditioned problem. It turned out to be quite hard to build a good, non memory-intensive and especially fast preconditioner that reduces the otherwise exorbitantly high iteration count. Without preconditioning the MATLAB routine `minres` was clearly the fastest solver within this group.

The search for good preconditioners finally led us to multigrid methods which are generally considered as being the fastest numerical methods for the solution of discretized elliptic partial differential equations. The implementation of a geometric multigrid solver for the given problem turned out to be a complex and cumbersome task. Therefore, we used the algebraic multigrid package PEBBLES from the Johannes Kepler University of Linz in Austria.

Algebraic multigrid is kind of a black box algorithm which is robust and handles discontinuities and anisotropies of the given problem. Although the use of algebraic multigrid saves us from implementing a smoother and a coarse grid correction scheme for geometric multigrid, it also requires us to accept the computational expensive setup phase which is about two-thirds of the algebraic multigrid solution time. Nevertheless, with PEBBLES we are now able to solve the problems of the FES-Tool about 20 times faster than before.

Given the restrictions of the field of application (Windows 2000, 32bit computer) it was not possible to apply this solver directly and NetSolve was used to leverage external computing power. Here again we have to accept a big trade-off in terms of transfer-time to an external 64bit computer but overall we achieved a speed up from previously used built-in MATLAB solvers to the current implementation from about 9 hours down to less than an hour.

With the current advancement in computer technology 64bit systems will become available more and more and the best performance would be achieved with a local 64bit workstation. To utilize this computer by the whole department it is again recommendable to access the solver remotely via NetSolve from a client running the FES-Tool. With a reasonable fast network connection the transfer time of the data within the department can be neglected and the server could be used for other tasks as well.

Using this method we have studied the problem itself and a model in more detail. Since the CT data is currently only available with a fixed dimension (314 × 266 × 51) it was useful to develop a smaller model problem that features similar properties than the original data but was much faster to solve. Using this model we discovered that scaling down within only one direction leads to just slightly better results than scaling down in all directions. Therefore, it would be more economical for the large data sets to not only scale down in z- but also in x- and y-direction, which would lead to much smaller systems.

Other properties of the problem like behavior of the relative residual and influence of anisotropy and conductivity of the electrode material were verified for the CT data and the new model problem.

Chapter 9

Outlook

With this work some achievements were obtained in the development of the FES-Tool to completely simulate the stimulation of denervated muscles in the thigh. However, many tasks remain and new questions arose.

As indicated in Chapter 3 it could possibly lead to smaller systems and higher accuracy when using the Finite Element method instead of the Finite Difference method. Through the modular structure of the FES-Tool it should be possible without to much changes to switch the mathematical model and keep the data input module, the solver and the analyzing tool (FES-Analyze). A first step in this direction could be to start with Model II of Chapter 7 and test it with FEMLAB [19].

For the solver itself there are two directions one can go from here. We have shown that multigrid methods are currently the fastest methods available and it would certainly make sense to stay on this path.

On the one hand the geometric multigrid approach could be implemented which would lead to the provably fastest solver possible for this system. Unfortunately, this would be quite a difficult task and also would rule out the general character the solver currently has.

On the other hand the algebraic multigrid algorithm could be refined or another package than PEBBLES could be used. Here, the software tool HYPRE (Chow et al. [15]) should be mentioned which is actively developed and promises to be one of the most advanced packages available, although aimed at parallel computers. Since there is now a Linux cluster available at the Department of Biomedical Engineering and Physics this would of course be one of the most fruitful paths by using a work-load distribution over several CPUs.

Finally, there is also a completely different approach for the given problem. Instead of calculating the electrical field within the thigh and using the results as basis where an activation will happen another way was proposed by Gene H. Golub. With a non-linear approach it should be possible to specify the regions where an activation should take place and together with the conductivity matrix this should lead to the optimal electrode configuration and stimulation parameters—see Winslow [59] as starting point.

Appendix A

Solver Configuration

This Appendix describes the use of the structure field `flag` which is used to configure the behavior of the solver. The available options are divided into 3 sections:

- solver: chooses the solver and its options,
- model: specifies which model to use and its configurations,
- auxiliary: allows debugging and automatically generated statistics.

Settings for the respective options are shown in parenthesis:

Solver

`solver` The solver that actually should be used, available are: (gs), (pcg), (bicg), (minres), (reoelm), (mg), (amg) and (superlu).

`precond` If the solver supports preconditioning you can choose here one of the available preconditioners: (none), (cholinc), (btdchol) or (amg).

`service` Choose if the problem should be solved (local), or remotely through (netsolve) or (harness).

`tol` Tolerance of the relative residual for iterative solvers.

`maxit` Maximum number of iteration for iterative solvers.

`type` Distinguishes between voltage (0) and current (1) sources.

`modifier` Transforms the equation system in various equivalent forms which is necessary for some solvers. Using `modifier`=2 will work for each solver.

`optionstr` Configuration options as comma separated string for PEBBLES.

`startvalue` If set to (1) the sparse vector `x` in `x.mat` is loaded and used as initial solution for iterative solvers.

Model

`problem` Chooses one of the 3 model problems Model I (p), Model II (m) or Model III (c) (see Chapter 7).

`size_x`, `size_y`, `size_z` Size of the model; when using CT data (Model III) this information is obtained from `configuration.mat`.

`complete_x`, `complete_y`, `complete_z` Original size of the data set.

`voxel_x`, `voxel_y`, `voxel_z` Size of a voxel.

`anisotropy_x`, `anisotropy_y`, `anisotropy_z` Values of anisotropy in the muscle.

`g_skin`, `g_fat`, `g_muscle`, `g_blood`, `g_bone`, `g_electrode`, `g_air` Conductivity values of the various tissue types.

`electrode_offset` Specifies the number of pixel the electrodes (point electrodes and electrode material) are moved away from their original position along the z-axis in longitudinal direction.

`conduct_str` A unique string characterizing the conductivity values (used as part of the filenames of `logfile` and `matrixfile`).

Auxiliary

`platform` Specifies for the logfile where the test was performed: Apollo (a), HP Itanium (h) or Simulant2 (s).

`logfile` Name of the logfile.

`matrixfile` Name of the file containing the model data.

`expand` If set to (1) the first and second derivative of the solution is calculated using the size of `complete_x`, `complete_y` and `complete_z`.

`dispstep` Shows every nth rel. residual in a log file of an iterative solver.

`debug` Choose a debugging level from (0) to (2).

`fastload` If set to (1) the model problems I and II are only created the first time and stored. On a second execution this model is loaded from the harddisk.

Appendix B

Curriculum Vitae

PERSONAL DATA

Full Name: DI Christoph Leopold Fabianek
born on July 21st, 1977 in Eggenburg; single; Austrian

Contact: fabianek@cs.utk.edu, +43 / 699 / 124 60 527 (mobile)

Address:
Mayerhofgasse 3 / 407
A-1040 Vienna, Austria
Tel: +43 / 1 / 505 53 84 – 407

Roseldorf 61
A-3714 Sitzendorf, Austria
Tel./Fax: +43 / 29 59 / 23 22

EDUCATION

since 2001: **Doctorate** @ Vienna University of Technology
Collaboration with the Vienna General Hospital
Research at the University of Tennessee / USA (Jan - May 2003)

1997-2001: **University** @ Vienna University of Technology
Master of Science in Technical Mathematics
Master Thesis about stock management
Tutor at the Dept. for Applied Math. and Numerical Analysis

1996-1997: **Military service** @ Vienna
Heeres-Datenverarbeitungsamt
(Data processing bureau of the Austrian Army)

1992-1996: **Vocational School** @ Klosterneuburg
HBLA u. BA fr Wein- und Obstbau
focus on viticulture and orcharding
finished with excellence

PROFESSIONAL EXPERIENCE

since 1998: **System Administration** @ Vienna
Carbone Lorraine GesmbH (international company)
Responsibilities: database maintenance & development, adoption to the Euro and Y2K for the ERP software

1994-2001: **Software for viticulture** @ Klosterneuburg
development of the first viticulture software under Windows
collaboration with Erbslöh (German beverage-technology company)

1999: **Security Management** @ Vienna
Specification of @-sign at APSS
(national infrastructure for digital signatures)

1997: **Financial Software** @ Parndorf
Development and completion of accounting software for a retail store

PUBLICATIONS

[1] C. Fabianek, W.N. Gansterer, C.W. Ueberhuber: *A Multi-Elimination Technique for Equilibrium Systems.* Technical Report AURORA TR2002-04, University of Vienna, 2002.

[2] C. Fabianek, W.N. Gansterer, C.W. Ueberhuber: *Experiments with a Multi-Elimination Technique for Equilibrium Systems.* Technical Report AURORA TR2002-05, University of Vienna, 2002.

[3] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, C. Fabianek, M. Miller, K. Sagi, K. Seymour, Z. Shi, S. Vadhiyar: *Users' Guide to NetSolve V2.0.* Innovative Computing Laboratory, University of Tennessee, Knoxville, 2003.

LANGUAGES

German (mother tongue), English (fluent), Japanese & French (basic)

INFORMATICS

Windows 9x, Windows NT, Linux
C, C++, VBA, Java, Perl, Pascal, Matlab, Oracle, HTML, MS Office, Navision

INTERESTS

sports (Karatedo, long distance running), philosophy, history of art

Vienna in December 2003

Bibliography

[1] *Cygwin User's Guide.* Red Hat, Inc., 2003.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen: *LAPACK Users' Guide*, 3rd edn. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.

[3] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, C. Fabianek, M. Miller, K. Sagi, K. Seymour, Z. Shi, S. Vadhiyar: *Users' Guide to NetSolve V2.0.* Innovative Computing Laboratory, University of Tennessee, Knoxville, 2003.

[4] W. E. Arnoldi: *The principle of minimized iteration in the solution of the matrix eigenproblem.* Quart. Appl. Math. (1951)(9), pp. 17–29.

[5] S. Atchley, S. Soltesz, J. Plank, M. Beck, T. Moore: *Fault-Tolerance in the Network Stack.* IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems (2002).

[6] W. Auzinger: *Numerik partieller Differentialgleichungen, eine Einführung.* Lecuter notes, TU Wien, 2003.

[7] R. E. Bank, C. C. Douglas: *Sharp Estimates for Multigrid Rates of Convergence with General Smoothing and Acceleration.* SIAM J. Numer. Anal. 22 (1984)(4).

[8] P. I. Barton, R. J. Allgor, W. F. Feehery: *DSL48S A Large-Scale Differential-Algebraic and Parametric Sensitivity.* Technical report, Department of Chemical Engineering Massachusetts Institute of Technology, 1997.

[9] M. Beck, J. Dongarra, G. Fagg, A. Geist, P. Gray, J. Kohl, K. M. M. Migliardi, T. Moore, P. P. ad S. Scott, V. Sunderam: *HARNESS: a next generation distributed virtual machine.* Future Generation Computer Systems 15 (1999), pp. 571–582.

[10] A. Brandt: *Multi-Level adaptive solutions to boundary-value problems.* Math. Comp. (1977)(31), pp. 333–390.

[11] T. Breyer: *Simulation of the 3-dimensional Electric Field in the Human Thigh Applied During Functional Electrical Stimulation.* Master's thesis, Vienna University of Technology, 2001.

[12] T. D. Cao, J. F. Hall, R. A. van de Geijn: *Parallel Cholesky Factorization of a Block Tridiagonal Matrix*, 2002.
`citeseer.nj.nec.com/509007.html`

[13] R. Carbunaru, D. Durand: *Toroidal Coil Models for Transcutaneous Magnetic Stimulation of Nerves.* IEEE Transactions on Biomedical Engineering 48 (2001)(4), pp. 434–441.

[14] T. Chandrupatla, D. Ashok: *Introduction to Finite Elements in Engineering.* Prentice-Hall, 1991.

[15] E. Chow, A. Cleary, R. Falgout: *Hypre User's manual, version 1.6.0.* Technical Report UCRL-MA-137155, Lawrence Livermore National Laboratory, Livermore, CA, 1998.

[16] E. Cuthill, J. McKee: *Reducing the bandwidth of sparse matrices.* Proc. 24th Nat. Conf. Assoc. Comput. Mach., ACM Publ. (1969), pp. 157–172.

[17] T. Davis, J. Gilbert, S. Larimore, E. Ng: *A column approximate minimum degree ordering algorithm*, 2000.

[18] G. Fagg, A. Bukovsky, J. Dongarra: *Harness and fault tolerant MPI.* In *Parallel Computing*, 2001.

[19] FEMLAB: http://www.femlab.com, 2003.

[20] R. Fletcher: *Conjugate Gradient Methods for Indefinite Systems.* In *Lecture Notes Math.*, Springer Verlag, Berlin-Heidelberg-New York, Vol. 506, 1976, pp. 73–89.

[21] I. Foster, C. Kesselman (eds.): *The Grid: Blueprint for a Future Computing Infrastructure.* Morgan-Kaufmann, San Francisco, CA, 1999.

[22] A. Geist, A. Beguelin, J. Dongarra, W. Liang, B. Manchek, V. Sunderam: *PVM: Parallel Virtual Machine - a User's Guide and Tutorial for Network Parallel Computing.* MIT Press, Cambridge, 1994.

[23] A. George, J. W. Liu: *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, New Jersey, 1981.

[24] G. H. Golub, C. F. Van Loan: *Matrix Computation*, 3rd edn. John Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, Maryland, 1996.

[25] B. Grotz: *Simulation der Funktionellen Elektrostimulation an denervierter Skelettmuskulatur im anisotropen 3-D Modell des menschlichen Oberschenkels.* Master's thesis, Fakultät für Elektrotechnik, Technische Universität Wien, 2002.

[26] W. Hackbusch: *Multigrid convergence theory*. Springer, Berlin, 1982.

[27] W. Hackbusch: *Multi-grid methods and applications*. Springer, 1985.

[28] D. Higham, N. Higham: *MATLAB Guide*. SIAM, 2001.

[29] A. L. Hodgkin, A. F. Huxley: *Resting and action potentials in single nerve fibres*. J. Physiol. 104 (1945), pp. 176–195.

[30] L. C. Junqueira, J. Carneiro, R. O. Kelley: *Histologie*. Springer-Verlag, 2002.

[31] H. Kern: *Funktionelle Elektrostimulation paraplegischer Patienten*. Öster. Z. Phys. Med. (1995)(1). Supplementum.

[32] F. Kickinger: *Algebraic multigrid for discrete elliptic second-order problems*. In *Multigrid Methods V* (W. Hackbusch, ed.), Springer Lecture Notes Comput. Sci. Eng. 3, Berlin, 1998, Proceedings of the 5th European Multigrid conference, pp. 157–172.

[33] C. Lanczos: *Solution of Systems of Linear Equations by Minimized Iterations*. J. Res. Natl. Bur. Stand. (1952)(49), pp. 33–53.

[34] J. Li: *Multifrequente Impedanztomographie zur Darstellung der elektrischen Impedanzverteilung im menschlichen Thorax*. Ph.D. thesis, Fakultät für Elektrotechnik und Informationstechnik, Universität Stuttgart, 2000.

[35] H. Lippert: *Lehrbuch Anatomie*. Urban & Fischer, 2000.

[36] J. W. H. Liu: *Modification of the minimum degree algorithm by multiple elimination*. ACM Trans. Math. Software (1985)(11), pp. 141–153.

[37] T. Mandl: *Segmentierungsmethoden zur Ermittlung der Ansiotropie der elektrischen Leitfähigkeit im menschlichen Oberschenkel (in progress)*. Master's thesis, Technische Universität Wien, 2004.

[38] J. Martinek: *Analyse der Aktivierung von denervierten Muskelfasern im menschlichen Oberschenkel bei Funktioneller Elektrostimulation*. Master's thesis, Institut für Analysis und Technische Mathematik, Technische Universität Wien, 2003.

[39] MATLAB: Application Program Interface Guide: MathWorks Inc., 1998.

[40] MATLAB: Application Program Interface Reference: MathWorks Inc., 1998.

[41] J. S. Plank, M. Beck, W. R. Elwasif, T. Moore, M. Swany, R. Wolski: *The Internet Backplane Protocol: Storage in the Network*. In *NetStore99: The Network Storage Symposium*, Seattle, WA, 1999.

[42] F. Rattay: *Electrical Nerve Stimulation: Theory, Experiments and Applications.* Springer Verlag, Wien, 1990.

[43] J. Reddy: *An Introduction to the Finite Element Method.* McGraw-Hill, 1984.

[44] M. Reichel: *Funktionelle Elektrostimulation denervierter Skelettmuskulatur.* Ph.D. thesis, Technische Universität Wien, 1999.

[45] S. Reitzinger: *Pebbles - User Guide.* Spezialforschungsbereich SFB F013, Linz, Austria, version: red edn., 2000.

[46] J.W. Ruge, K. Stüben: *Algebraic Multigrid (AMG).* SIAM, Frontiers in Applied Mathematics 5 (1986), pp. 73–130.

[47] Y. Saad: *Iterative methods for sparse linear systems.* PWS Publishing Companies, Boston, Mass., 1996.

[48] Y. Saad, M.H. Schultz: *GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems.* SIAM J. Sci. Statist. Comput. (1986)(7), pp. 856–869.

[49] H.R. Schwarz: *Numerische Mathematik.* B.G. Teubner, Stuttgard, 1986.

[50] M. Solomonow: *External control of the neuromuscular system.* IEEE-Trans. Biomed. Eng. BME-31 (1984), pp. 752–763.

[51] P. Sonneveld: *CGS: A fast Lanczos-type Solver for Nonsymmetric Linear Systems.* SIAM J. Sci. Statist. Comput. (1989)(10), pp. 36–52.

[52] B. Stroustrup: *The C++ Programming Language.* Addison-Wesley, 200.

[53] U. Trottenberg, C. Oosterlee, A. Schüller: *Multigrid.* Academic Press, 2001.

[54] P. Vanek, J. Mandel, M. Brezina: *Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems.* Technical Report UCD-CCM-036, 1995.
`citeseer.nj.nec.com/vanek96algebraic.html`

[55] C. Vinschen, C. Faylor, D. Delorie, P. Humblet, G. Noer: *Cygwin User's Guide.* Red Hat, 1998. Http://cygwin.com/cygwin-ug-net/cygwin-ug-net.html.

[56] H.A. van der Vorst: *BI-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Non-Symmetric Linear Systems.* SIAM J. Sci. Statist. Comput. (1992)(13), pp. 631–644.

[57] H. A. van der Vorst: *Iterative Krylov Methods for Large Linear Systems.* Cambridge University Press, 2003.

[58] T. Washio, C. Oosterlee: *Flexible multiple semicoarsening for three-dimensional singularly perturbed problems.* SIAM J. Sci. Comput. (1998)(9), pp. 1646–1666.

[59] A. M. Winslow: *Numerical solution of the quasilinear Poisson equation in a nonuniform triangle mesh.* J. Computational Phys. 1 (1966-67)(149).

[60] I. Yavneh: *On red-black SOR smoothing in multigrid.* SIAM J. Sci. Comput. (1998)(17), pp. 180–192.