Step-size control and corrector methods in numerical continuation of ocean circulation and fill-reducing orderings in multilevel ILU methods

Arie de Niet

Department of Mathematics

RuG

Master's Thesis

Step-size control and corrector methods in numerical continuation of ocean circulation and fill-reducing orderings in multilevel ILU methods

Arie de Niet

Supervisor:
Dr.ir. F.W. Wubs
Department of Mathematics
University of Groningen
P.O. Box 800
9700 AV Groningen

August 2002

Abstract

The equations of the thermohaline ocean circulation model are complex and contain a large number of physical constants and parameters. For stability analysis of the equations a numerical continuation program is used, that is based on the predictor-corrector method. We examine several possible improvements of the continuation code.
The standard corrector is the Newton method. However the Newton iterations are expensive due to the solution of a large linear system. We test the performance of several alternative correctors. The adaptive Shamanskii method, a combination of the Newton and Newton-chord method, appears to be the fastest corrector. We also develop a method in order to adapt the step size along the branch automatically. The implementation of adaptive Shamanskii and step size control in the continuation code causes a considerable speed up of about a factor 4.
In the corrector iterations large sparse linear systems of saddle point type are to be solved. The solver in the present code is MRILU. We present some alternative solvers from literature for the saddle point equation and do some convergence analysis on them.
Finally we examine the possibility of incorporation of fill-reducing ordering ideas in multilevel ILU methods. We are able to construct a useful ILU factorization based on a nested-dissection ordering. The preconditioner is compared with MRILU for the Poisson and Stokes equation.

Contents

Chapter 1

Introduction

This report considers several improvements of a numerical continuation code for the three-dimensional thermohaline ocean circulation. The code is developed at the Institute for Marine and Atmospheric research Utrecht (IMAU) by the research group on ocean circulation and climate. The objective for the program is the computation of the ocean circulation flow under changing circumstances. The ocean flow is influenced by a lot of factors like earth rotation, wind field and temperature distribution. The governing equations contain also a lot of physical parameters. A small change in one or more factors might have a large impact on the ocean circulation. For instance a slightly higher global temperature leads to climate changes on earth.

In the first part of this report we introduce numerical continuation that is based on a predictor-corrector method. To improve the code we focus on two important aspects: the choice of the step size and the corrector. In Chapter 3 we propose a method for automatic step size control and in Chapter 4 we discuss a few correctors. Step-size control and the correctors are tested for the Bratu problem and an ocean circulation problem. We will show that the implementation of the adaptive Shamanskii method as corrector in combination with step-size control results in a considerable speed-up.

The research in the second part is still far from direct implementation in the continuation code. There we focus on the system solver. In the process of numerical continuation we have to solve large sparse systems of equations. The present code uses MRILU to do this. MRILU shows good performance on many problems, but on Stokes-like problems, that play an important role in all flows, it produces a relatively high level of fill (the number of non-zeros) in the preconditioner.
We searched in literature for alternative solving techniques. A brief overview of the relevant solvers is presented in Chapters 7 and 8. On a few methods we did some convergence analysis. Furthermore we examine in Chapter 9 the possibility of incorporation of fill-reducing orderings in multilevel ILU methods like MRILU. We construct an incomplete LU factorization based on a nested-dissection ordering. Unless several non-essential simplifying restrictions the preconditioner can compete with MRILU and looks promising.

Part I

Chapter 2

Numerical Continuation

In this chapter we introduce numerical continuation. We explain the principles in a few sections. For a more extended introduction we refer to [Sey94, chapter 4] and [AG90]. At the end we point out some gaps in the continuation process: choices that are still to be made or parts that allow improvements. All remaining chapters of this thesis are devoted to research on these gaps.

2.1 Basic Principles of Numerical Continuation

Assume that we have to solve an equation $f(u) = 0$ and that this equation contains some parameter μ. For example this parameter can be the Reynolds number in the discretized Navier-Stokes equations. In general the solution of the equation depends on the value of this parameter. Therefore we rewrite the equation such that it expresses this dependence

$$f(u, \mu) = 0. \tag{2.1}$$

If the dimension of u is d and if f is a d-dimensional vector function, this is a system of d equations and $d+1$ unknowns. So the equation defines a one-dimensional curve in the parameter space $\mathbb{R}^{d+1}$. This curve is a branch of solutions that gives the dependence of the solution on the parameter μ. The computation of this branch is a *continuation* problem.

By continuation we can find the relation between the parameter and the solution. It gives information about the effect of disturbances in the parameter, the stability of the solution and whether there is a solution at all for each parameter value. On the branch we can encounter bifurcations, that is qualitive changes in the behavior of the solution, such as turning points, pitchfork and Hopf bifurcations. Following the branch and jumping over the bifurcations we can find out whether there is more than one solution for a parameter.
Continuation is also useful for complex problems with many problem parameters. If there is a solution known for some set of parameter values, we can use this solution as a starting point for the continuation process. We can follow the paths of all parameters until we reach the desired value for each parameter.
In case of the ocean circulation problem, continuation of steady states is used for a combination of the reasons mentioned. First the problem is huge and contains many problem parameters like earth rotation speed, the atmospheric temperature function, the precipitation, the wind field, the shape of the continents, the shape of the bottom of the ocean and a lot of physical constants. Stability analysis and the determination of multiple solutions is also an important reason.

We are not interested in all points on the continuation curve. It is enough to find a few solutions to get an impression of the behavior of the solution. We try to walk through the curve in a few steps. This process is called *numerical continuation*.

If s is the parameterization variable, we can add an equation $q(u, \mu, s) = 0$ to equation (2.1). This extra equation characterizes the parametrization. Now we have to solve the system

$$g(x) \equiv \begin{bmatrix} f(u,\mu) \\ q(u,\mu,s) \end{bmatrix} = 0, \text{ with } x = \begin{bmatrix} u(s) \\ \mu(s) \end{bmatrix}. \tag{2.2}$$

For each value of s this system has $d+1$ equations and $d+1$ unknowns.

Before we can use numerical continuation, we have to choose the parameterization for the curve. A parameterization identifies the solutions on the branch. For this parameterization several methods are available, however some are more useful than others.
The most obvious choice is using the equation parameter μ as curve parameter too, so $q = \mu - s$. But problems will occur at the turning points of the curve, where we have $\frac{du}{d\mu} = \infty$. If we don't have much information about the curve, we want a method that is able to handle turning points and other bifurcations. This is possible with the pseudo-arclength parameterization, that is described in the next section.

Finally we have to find a method to get from one point on the curve to the next one. We use a predictor-corrector model that is explained in Section 2.3.

2.2 Pseudo-Arclength Parameterization

For the parameterization of the curve we use its arclength. So let s be the arclength parameter, that is implcitly defined by

$$\zeta \left\| \frac{du}{ds} \right\|_2^2 + \left(\frac{d\mu}{ds} \right)^2 = 1 \tag{2.3}$$

for $\zeta = 1$. The tuning factor $0 < \zeta < 1$ belongs to the pseudo-arclength parameterization and can be used to place different emphasis on u and μ. We choose $\zeta = 1/d$, the inverse of the dimension of u.

Assume we have for $s = s_0$ a solution on the branch $(\bar{u}, \bar{\mu}) = (u(s_0), \mu(s_0))$, that will serve as initial solution. We want to know the solution on arclength distance Δs. Let $s_1 = s_0 + \Delta s$, then the next solution will be $(u, \mu) = (u(s_1), \mu(s_1))$. These two points can be used to replace the derivatives in equation (2.3) by the following backward difference approximations

$$\frac{du}{ds} = \frac{u - \bar{u}}{\Delta s} + O(\Delta s), \quad \text{and} \quad \frac{d\mu}{ds} = \frac{\mu - \bar{\mu}}{\Delta s} + O(\Delta s). \tag{2.4}$$

Substitution of (2.4) in equation (2.3) and a multiplication with $(\Delta s)^2$ gives the desired extra equation

$$q(u, \mu, s) = \zeta \|u - \bar{u}\|_2^2 + (\mu - \bar{\mu})^2 - (\Delta s)^2. \tag{2.5}$$

2.3 Predictor-Corrector Method

In the numerical continuation algorithm we use a predictor-corrector method. That means, given an initial point we predict somehow the next point on the branch. Probably this prediction is wrong, so we will have to correct it. Commonly used is *Euler-Newton* continuation, that has forward Euler as predictor and Newton as corrector. The Euler predictor provides the first guess

$$(u_0, \mu_0) = (\bar{u}, \bar{\mu}) + \Delta s \cdot (\dot{\bar{u}}, \dot{\bar{\mu}}), \tag{2.6}$$

where $(\dot{\bar{u}}, \dot{\bar{\mu}})$ denotes the derivative of $(\bar{u}, \bar{\mu})$ with respect to s. This value $x_0 = (u_0, \mu_0)$ is corrected with a Newton method, for $n = 1, 2, \ldots$

$$g'(x_n) \cdot \Delta x = -g(x_n), \tag{2.7}$$

$$x_{n+1} = x_n + \Delta x, \tag{2.8}$$

until the convergence criterion is satisfied: $\|\Delta x\|_\infty < \varepsilon$.

For the computation of the first guess in equation (2.6) we need to know $\dot{\bar{u}}$ and $\dot{\bar{\mu}}$. It is possible to compute both, but in Section 2.5 we will see that we have to solve a large system, which is quite expensive, even if we have a good preconditioner. It is much easier to use the last two points on the branch to determine an approximation of the derivative. Therefore we use the same backward difference approximation for $(\dot{\bar{u}}, \dot{\bar{\mu}})$ as we did for $(\dot{u}, \dot{\mu})$ in equation (2.4). However at the initial point, we still have to compute the real derivative.
With the backward difference approximation for $(\dot{\bar{u}}, \dot{\bar{\mu}})$ we actually don't use an Euler predictor but a *secant predictor*. Both methods are of first order in Δs, so the choice doesn't really affect the quality of the prediction.

2.4 Computations in the Newton Process

In each iteration of the Newton process we have to solve an equation with the Jacobian $g'(x_n)$, that can be written as

$$g'(x) = \begin{bmatrix} \Phi & f_\mu \\ (\frac{\partial q}{\partial u})^T & \frac{\partial q}{\partial \mu} \end{bmatrix}, \tag{2.9}$$

where Φ is the matrix of derivatives of f with respect to u, and f_μ the vector of derivatives with respect to μ. Note that all parts of the matrix are dependent of u, μ and s. The derivatives of q can be obtained by differentiating equation (2.5). If they are inserted in the Jacobian, we have to solve the following system in each iteration step

$$\begin{bmatrix} \Phi & f_\mu \\ 2\zeta \cdot (u - \bar{u})^T & 2 \cdot (\mu - \bar{\mu}) \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta \mu \end{bmatrix} = \begin{bmatrix} -f \\ -q \end{bmatrix}. \tag{2.10}$$

The solution of this system can be obtained in two steps in which only linear systems with Φ are solved. Expanding (2.10) gives

$$\Phi \cdot \Delta u + f_\mu \cdot \Delta \mu = -f, \tag{2.11}$$

$$2\zeta \cdot (u - \bar{u})^T \cdot \Delta u + 2 \cdot (\mu - \bar{\mu}) \cdot \Delta \mu = -q. \tag{2.12}$$

Now we introduce z_1 and z_2, that are solutions of

$$\Phi z_1 = -f, \quad \text{and} \quad \Phi z_2 = f_\mu. \tag{2.13}$$

Inserting z_1 and z_2 in equation (2.11) gives

$$\Phi \cdot \Delta u + \Delta \mu \cdot \Phi \cdot z_2 = \Phi \cdot z_1, \quad \text{so} \quad \Delta u = z_1 - \Delta \mu \cdot z_2. \tag{2.14}$$

Now Δu can be eliminated from (2.12). That results in an expression for $\Delta \mu$

$$\Delta \mu = \frac{-q - 2\zeta \cdot (u - \bar{u})^T z_1}{2 \cdot (\mu - \bar{\mu}) - 2\zeta \cdot (u - \bar{u})^T z_2}. \tag{2.15}$$

Summarizing, each step in the Newton iteration starts with the computation of z_1 and z_2 by solving (2.13). Then $\Delta \mu$ and Δu can be computed with equation (2.14) and (2.15). The solution of the two systems with Φ is the most expensive step in this process. Fortunately we can use a preconditioner for this matrix twice. Table 2.1 contains the operations that are performed chronologically in each Newton iteration.

2.5 Computing the First Tangent

At the initial point on the branch we cannot use the secant predictor, because there is no previous point, so we have to use the Euler predictor. For the Euler predictor we have to compute the

1.	*Construction of righthand side* $-f$
2.	*Numerical computation of derivative* f_μ
3.	*Arclength parameterization: computation of* $q(u, \mu, s)$
4.	*Construction of Jacobian matrix,* Φ
5.	*Construction of preconditioner for* Φ
6.	*Solution of system* $\Phi z_1 = -f$
7.	*Solution of system* $\Phi z_2 = -f_\mu$
8.	*Computation of* q_u *and* q_μ
9.	*Computation of* Δu *and* $\Delta\mu$
10.	*Updates,* $u := u + \Delta u$, $\mu := \mu + \Delta\mu$
11.	*Approximation of Newton error:* $\|\Delta u, \Delta\mu\|_\infty$

Table 2.1: List of operations performed in each Newton iteration.

tangent $(\dot{u}, \dot{\mu})$, the derivative of a point on the curve to the arclength parameter. The derivative can be obtained by differentiating the equation $f(u, \mu) = 0$ with respect to μ, which gives

$$\Phi \frac{du}{d\mu} + f_\mu = 0.$$

Rearrangement of the equation results in

$$\Phi \frac{du}{d\mu} = -f_\mu. \tag{2.16}$$

As before we have to solve a linear system with Φ. Now the derivatives of u and μ with respect to s can be obtained from the pseudo-arclength equation. Substituting

$$\frac{du}{ds} = \frac{du}{d\mu} \cdot \frac{d\mu}{ds} \tag{2.17}$$

into equation (2.3) and rewriting the expression we obtain

$$\frac{d\mu}{ds} = 1/\sqrt{1 + \zeta \left\| \frac{du}{d\mu} \right\|^2}.$$

Finally, using equation (2.17) we find $\dot{u}$ and we have the direction of the curve at the initial point. The major disadvantage is the solution of the system with Φ in equation (2.16). However we have to do this only once at the beginning of the continuation process.

2.6 Possible Improvements

There are several possibilities to improve the algorithm for numerical continuation as sketched in this chapter. For the special case of thermohaline ocean circulation Levine [Lev01] has made some improvements in his master's thesis. He chooses for example smart initial values for z_1 and z_2 and tunes the accuracy of the solver for both vectors.
We examine improvements in three parts of the algorithm.

step size We have to determine an appropriate step size Δs. If it is too large we get troubles in the Newton process, if it is too small we take unnecessarily many steps. Until now the step size is chosen manually and fixed for several steps. In the next chapter we develop a method that adapts the step size automatically.

corrector In Section 2.3 we introduced a Newton process as corrector. However this brings out an expensive calculation of a preconditioner for Φ in each Newton step. In Chapter 4 we discuss some alternatives for the Newton method.

solver In the present THCM program the preconditioner for Φ is constructed by MRILU. But is MRILU really the best solver? In part II of this thesis we do some analysis on the problem and examine the alternatives for the MRILU solver.

Chapter 3

Step-Size Control

As input for the THCM program one gives, among other parameters, the step size (Δs) for the first step in the continuation algorithm. This step size is not altered during the run of the program, so the next steps will have the same size. That is disadvantageous because the smooth parts of the curve will allow a much larger step size than the difficult parts as near turning points or pitchfork bifurcations.
If the step size is too large for a certain part of the curve, the Newton process will not converge and the program terminates. In a restart one can choose a smaller step size and hope that the program will run through the difficult part with this smaller step size. This manual adjustment of step size is quite laborious and therefore an automatic step size adjustment is desirable.

We are mainly interested in the point(s) on the curve where the parameter equals the target value $\hat{\mu}$, so we would like to walk as fast as possible through the branches until we reach this value. We don't want to take unnecessarily small steps. But if the steps are too large we risk very slow convergence in the Newton process. Therefore the step size has to be chosen such that on one hand the step size in the predictor is as large as possible, while on the other hand the number of iterations in the corrector is as small as possible. These demands are conflicting, so we have to find the balance between both. In this chapter we propose a method that tries to find this balance.

3.1 An Optimal Number of Iterations

If the step size is too small, we need just a few Newton iterations, but we will have to take a lot of steps and the branch tracing will be costly. On the other hand if the step is too large, we will need a lot of Newton iterations to correct the prediction and again the branch tracing will be costly. We have to stay somewhere in between, so Seydel [Sey94, paragraph 4.6] suggests there exist something like an *optimal number of iterations*, call it N_{opt}. Let N be the number of iterations in the last Newton process, if $N > N_{opt}$ we could better take a smaller step size and if $N < N_{opt}$ we may be able to take a larger one.
To determine how much the step size becomes larger or smaller we introduce the following number

$$\xi = \frac{N_{opt}}{N}.$$

If we multiply Δs after each Newton iteration with this number, the step size becomes larger if Newton converges within N_{opt} iterations and smaller if it requires more than N_{opt} iterations.
The step size should not change too rapidly, therefore we introduce the real multiplier $\bar{\xi}$, that

forces the step size to change with at most a factor two,

$$\bar{\xi} = \begin{cases} 0.5 & \text{if} \quad \xi < 0.5 \\ \xi & \text{if} \quad 0.5 \leq \xi \leq 2 \\ 2 & \text{if} \quad 2 < \xi \end{cases} .$$

Now we can use $\bar{\xi}\Delta s$ as step size in the next continuation step.

We still have to choose the optimal number of iterations. In case of Newton continuation it should be near 5, however the method is not very sensitive to the precise value of N_{opt}, as long as it is reasonably large. The optimal number of iterations depends on the type of corrector that we use. So if we choose another corrector, we may have to choose another value for N_{opt} too.

3.2 Trial and Error

If the method as proposed in the previous section is used as stand-alone-method for step length control, it works well as long as the difficulties on the branch do not occur too suddenly. Unfortunately on the flat parts of a continuation curve the step size can become relative large and then bifurcations occur quite sudden. So we have to find a way to adapt the step size in the neighborhood of such points.
An easy way is to impose a maximum step size that fits the difficult parts. This has as main disadvantage that the maximum step size has to be relative small, which means that the step size is limited at the smooth parts of the curve, whereas we would like to take there the largest steps possible. Another drawback is that it is never certain that the maximum step size is small enough to handle the most difficult parts, if one has no information about the curve on beforehand.

There is a much better solution of this problem, namely the use of *trial and error*. That means, we calculate the size of the next continuation step, we let the predictor guess the next point on the curve and if the corrector fails to converge due to a sudden bifurcation, we simply try it again with a step that is half as large as the original step size. The price of being able to take large steps on smooth parts is the possible need to take steps again if the corrector fails.
The extra costs of a few failing steps should be reduced as much as possible. Therefore we would like to recognize a failing corrector as soon as possible. Happily there are many indications for bad convergence. The process can be aborted if the corrector diverges, exceeds a maximum number of iterations (for example $2 \cdot N_{opt}$) or exceeds a time limit.

In chapter 5 we test this approach for the step size in combination with the one we presented in the previous section.

Chapter 4

A Few Corrector Methods

In the continuation process based on the predictor-corrector method we have to solve an equation like

$$g(x) = 0,$$

with a corrector method, where the predictor provides us an initial guess. In the original THCM code the Newton method is used, however there may be other methods that serve better. In this chapter we present a brief overview of alternative corrector methods.
There is a wide range of methods available to solve an equation as above. For an extended overview we refer to [Bre97] and [Lev01, chapter 6]. We will consider just a few correctors that are closely related to the Newton method. The main reason for this restriction is that these variants can be easily implemented in the present code.

In this chapter we assume that g' is Lipschitz continuous in the neighborhood of x, that means there exist some constant $L > 0$ such that $\|g'(x) - g'(y)\| < L \cdot \|x - y\|$. All norms are Euclidean norms.

4.1 Newton Method

We have already introduced the Newton method in equations (2.7) and (2.8). If we merge these equations, we get the following relation between x_n and x_{n+1}

$$x_{n+1} = x_n - [g'(x_n)]^{-1} \cdot g(x_n).$$

Most important is the inverse of $g'(x_n)$, that has to be computed in each iteration. Especially for large systems this can be very expensive. Main advantage is the *quadratic* convergence behavior of the Newton method, that means

$$\lim_{n \to \infty} \frac{\|x_{n+1} - x\|}{\|x_n - x\|^2} = k$$

for some real constant $k > 0$.
So if we have an initial solution x_0 that is close enough to x, the Newton method will converge in a few steps to a solution with the desired accuracy.

4.2 Newton-Chord Method

To avoid the expensive computation of the inverse of a large matrix in each iteration step, we can use a *parallel chords method*, that has the following iterative rule

$$x_{n+1} = x_n - A^{-1} \cdot g(x_n),$$

for some matrix A, that we will choose later.
With this rule we still have to do the expensive computation of an inverse, namely that of the matrix A, but we have to do it just once and we can use the explicitely computed inverse or (incomplete) LU factorization in all iteration steps.
Now the convergence behavior becomes *linear*, that means

$$\lim_{n\to\infty} \frac{\|x_{n+1} - x\|}{\|x_n - x\|} = k$$

for some real constant $k > 0$. Because of the lower rate of convergence a parallel chords method will need more iterations than the Newton method.

The most obvious choice for the matrix A in the parallel chords method is

$$A = g'(x_0),$$

which makes the first step equal to the first step of the Newton method. The parallel chords method with this choice for A is also called the *Newton-chord method.*

4.3 Shamanskii Method

The *Shamanskii method* alternates iterations with the Newton method and the Newton-chord method. This mixing results in a sequence of outer iterations existing of one step with the Newton method and inner iterations existing of m steps with the Newton-chord method. This can be summarized in the following iterative procedure

$$\begin{aligned} y_1 &= x_n - [g'(x_n)]^{-1} \cdot g(x_n) \\ y_{j+1} &= y_n - [g'(x_n)]^{-1} \cdot g(y_j) \quad \text{for} \quad 1 \le j \le m-1 \\ x_{n+1} &= y_m \end{aligned}$$

With $m = 1$ we obtain the original Newton method and with $m = \infty$ we get the Newton-chord method.

In comparison with Newton the Shamanskii method reduces the number of outer iterations considerably. Taking a few inner iterations after each Newton step, speeds up the convergence and we have to compute less expensive matrix inverses.
It can be proven that under reasonable circumstances the sequence x_n converges to x super linearly with order $\alpha > 2$, which means that for some real constant $k > 0$,

$$\lim_{n\to\infty} \frac{\|x_{n+1} - x\|}{\|x_n - x\|^\alpha} = k.$$

For the inner iterations (y_n) the convergence is super linear with $1 < \alpha < 2$. So the convergence speed lies between that of Newton-chord and Newton.

4.4 Adaptive Shamanskii method

We can slightly change the Shamanskii method by not choosing m beforehand, but to determine it during the iterative process. The basic idea behind the *adaptive Shamanskii method* is simple: if Newton-chord doesn't converge fast enough we have to take an extra Newton step. The switch between Newton and Newton-chord and back can take place at every moment of the iterative process.
However we have to determine when we better take a single expensive Newton step instead of a number of cheap Newton-chord steps. Therefore we introduce T_o, the time needed for an outer

iteration, and T_i, the time needed for an inner iteration. Let further k_n be the convergence constant of the quadratic Newton method and k_{nc} the convergence constant of the linear Newton-chord method.
If $\varepsilon_n = \|x_n - x\|$ is the error of the n-th approximation of x, then with the Newton method the next approximation will have an error close to $\varepsilon_{n+1} = k_n \varepsilon_n^2$. The number of Newton-chord iterations, needed to attain the same accuracy, can be calculated from the equation $\varepsilon_{n+1} = (k_{nc})^l \varepsilon_n$, that gives

$$l = \log(k_n \varepsilon_n) / \log(k_{nc}).$$

The amount of time needed by l inner iterations is $l \cdot T_i$. As long as $l \cdot T_i < T_o$, the time for the inner iterations is less than the time for one outer iteration, so it is beneficial to perform the inner iterations. The relation $l \cdot T_i < T_o$ can be rewritten such that we get the relation

$$k_{nc} < \exp\left[\log(k_n \cdot \varepsilon_n) \cdot (T_i / T_o)\right]. \tag{4.1}$$

After each inner iteration we can calculate new estimates for k_{nc} and T_i and check if this relation still holds. If it doesn't we perform an outer iteration that also gives new estimates for k_n and T_o. The error ε_n is approximated by $\|f(x_n)\|_\infty$.

4.5 Discussion

Which of the presented methods is best as corrector depends on several aspects of the continuation problem. An important role plays (i) the quality of the initial guess and the required accuracy, (ii) the relative time spent in the construction of the Jacobian, (iii) the relative time spent in the solver. In case of a bad initial guess or high accuracy Newton method is best used, because Newton-chord will require too many steps. If the time spent in the solver is the bottleneck one better uses the Newton-chord methods, with its cheap iteration steps.
Amongst the above methods the adaptive Shamanskii method cleverly uses information about the different aspects to determine what method, Newton or Newton-chord, is best used at each moment. The adaptive Shamanskii method can be implemented easily and gives theoretically the fastest convergence in time possible. Therefore that method is expected to be superior to the others.

Chapter 5

Two Test Problems

In the previous chapters we presented a method to control the step size and we introduced several correctors. The equation of step size control (3.1), that is used to calculate the size of the next step, contains an optimal number of iterations. The value of that number depends on the corrector we use. The Newton method should converge in less steps than the Newton-chord method, so the N_{opt} for Newton should be smaller than the N_{opt} of Newton-chord.
There is some trade off between the corrector and the step size control. The convergence in the corrector determines the next step size, but also the step size determines the distance to the next prediction and that influences the convergence in the corrector. Because of the close relation we test both step size and corrector in one test.

We introduce two test problems in this chapter: the Bratu problem and an ocean circulation problem. The first is used because it is a standard continuation problem, containing flat branches and a turning point. The second is an artificial problem of ocean circulation, that is chosen because it contains some typical difficulties of numerical ocean modeling, the area of the direct application of our methods. After a short description of both problems we present the results of step size control and the comparison of correctors.

5.1 The Bratu Problem

The first test problem is the two-dimensional Bratu problem, that is defined on the area Ω, the unit square in $\mathbb{R}^2$. The differential equation of the Bratu problem contains a parameter λ, that will serve as continuation parameter. We have the following class of nonlinear equations

$$-\Delta u - \lambda e^u = 0 \quad \text{in } \Omega, \tag{5.1}$$

with the Neumann boundary condition $u = 0$ on $\partial\Omega$. The two-dimensional Bratu problem has some physical background and is also know as the *solid fuel ignition problem*. It is a special case ($N = 2$) of the classical Liouville-Gelfand problem, that also has equation (5.1), but the area is defined $\Omega \subset \mathbb{R}^N$ with $N > 0$.

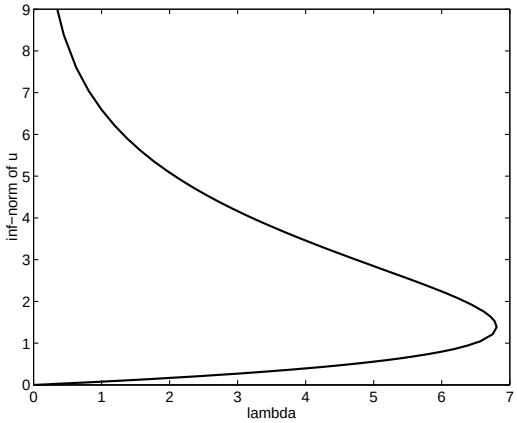

Figure 5.1: Bifurcation diagram of the Bratu problem.

The solution of equation (5.1) depends on the value of λ. In general the solution is a 'hill', that is symmetric along all symmetry axes of the unit square and reaches its highest value in the center point. $\|u\|_\infty$ is the largest value over all grid points and therefore equals the value in the center point. In Figure 5.1 this value is plotted against the

parameter λ, which results in the bifurcation diagram of the Bratu problem. Note that the curve contains a turning point, that is supposed to cause difficulties for the step size control and the convergence in the corrector. The turning point occurs at $\lambda_{FK} \approx 6.81$, the Frank-Kaminetskii parameter. For $0 < \lambda < \lambda_{FK}$ there are two solutions for each λ.
An initial solution for the continuation process is obtained easily. If $\lambda = 0$ the equation becomes $\Delta u = 0$ with the trivial solution $u = 0$. So $(u, \lambda) = (0, 0)$ serves as initial point.

We have written a program for the numerical continuation process of the Bratu problem using MATLAB. The program contains all steps of the continuation process as presented in Chapter 2. For the solution of a system with the Jacobian matrix Φ we use a simple LU factorization provided by a tool of MATLAB.
For equation (5.1) a finite difference discretization on a 31×31 grid is used. So the number of unknowns is 961.

5.2 The Ocean Circulation Problem

The second problem has its origin in a three-dimensional thermohaline ocean circulation model. The variables in the ocean model are the velocity in all three directions $u = (u_1, u_2, u_3)$, the pressure p, the salinity S and the temperature T. The relations between these variables are given in a number of complex equations. For an extended description of the ocean model, the involved equations and linearizations we refer to [DOWB01]. For now it is enough to know that the equations contain some temperature coefficient η_T, that controls the amplitude of the prescribed atmospheric temperature. This coefficient induces nonlinear behavior and will serve as continuation parameter. The target value for the parameter is 10, where the amplitude of the atmospheric temperature obtains its real value.

We test the ocean circulation problem for a box in the North-Atlantic ocean with $12 \times 28 \times 6$ grid points. In each grid point we have 6 variables, so we have a total of 12096 unknowns. For the exact size of the box and the values of all other parameters see appendix A.
Figure 5.2 contains the bifurcation diagram of the ocean circulation problem with on the horizontal axis the temperature coefficient and on the vertical axis the maximum of the meridional overturning stream function. The overturning stream function is a non dimensional quantity and indicates the volume transport. The maximum of the overturning stream function is a dimensional number expressed in Sverdrups (Sv).

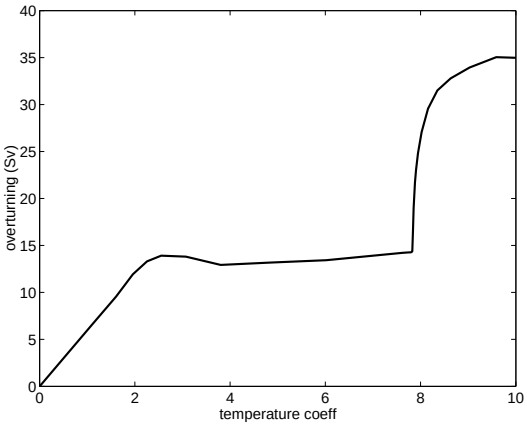

Figure 5.2: Bifurcation diagram of the ocean circulation problem.

The curve of the ocean circulation problem contains a bifurcation point. If $\eta_T \approx 7.9$ some kind of pitchfork bifurcation appears. The continuation algorithm always follows the left branch of the pitchfork.

The group of Dijkstra at the IMAU in Utrecht has written a program for the numerical simulation of thermohaline ocean circulation (see [DOWB01]). The program is called THCM and is written in the language FORTRAN. It was sufficient to add and adapt some subroutines for step size control and switching of corrector method.

5.3 Results for Bratu

In the test for the Bratu problem we compare three different correctors: Newton, Newton-chord and the adaptive Shamanskii, as described in the previous chapter. For the step size control we have to choose a value for N_{opt}, the optimal number of iterations. We tested for all methods several values for N_{opt} and it appeared that the best choice is $N_{opt} \in \{3, 4, 5\}$ for the Newton method and $N_{opt} \in \{6, 7, 8, 9\}$ for the other methods. As long as N_{opt} is chosen within these ranges, the actual choice for N_{opt} will not really influence the results. For the test we choose $N_{opt} = 4$ in case of the Newton method and $N_{opt} = 7$ in case of the other correctors.

To make sure that the continuation process follows the right branch, we set the maximum step size to 2. This prevents too large step sizes and strange jumps after the turning point. Without a maximum step size 'branch jumping' can occur, that means the continuation process jumps back to the lower branch, while it is supposed to follow the upper one. Just a few times on the flat branches the maximum step size is really needed. Therefore it hardly influences the total work and time, so the comparison is still fair.
We are just interested in the behavior of step size control and correctors in the neighborhood of the turning point. Therefore we stop the continuation process as soon as it passes the value $\lambda = 2$ for the second time.

We compare the quality of the correctors and the step size control by running the continuation process for each corrector with several initial step sizes. The initial step size has a relative large impact on the results. It determines at what point the continuation algorithm enters the difficult branches. This can make a difference of 30 percent of total time. For a fair comparison we have to neutralize the effect of the initial step size. We do this by averaging the results over seven runs with initial step sizes from 0.7 to 1.3.
In each run we count the number of continuation steps, failing steps (the ones that do not converge), (incomplete) LU factorizations and solved systems. We also measure the total time needed to follow the branch until we reach the target value. Table 5.1 contains an overview of all measured quantities and the abbreviations that are used in the other tables.

step :	number of continuation steps
fail :	number of failing steps (no convergence of corrector)
(I)LUs :	number of (incomplete) LU factorizations
sols :	number of solves of a (preconditioned) system
time :	total time needed to follow the branch

Table 5.1: Measured quantities.

The results of each corrector after averaging over the initial step size can be found in Table 5.2. Appendix B contains the results of the runs for all individual step sizes.
If we look at the total solve time, the best corrector appears to be the adaptive Shamanskii method, but the difference with the Newton-chord method is small. The two methods beat the original Newton method by far. The time needed to follow the branch is reduced by a factor two. This considerable gain of time is caused mainly by avoiding expensive LU factorizations (LUs).
The adaptive Shamanskii is constructed such that in general it is faster than Newton-chord. The experimental results confirm this superiority. By a few extra Newton steps, the number of systems to be solved is reduced considerably, almost a factor two. So the extra costs of the LU factorization is sufficiently compensated by the reduced costs in the system solver.
The times needed to construct an LU factorization (T_{LU}) or to solve a system (T_{sol}) are expected to be approximately constant. Therefore we have the following formula for the total costs of following the branch

$$T_{total} = \text{LU} \cdot T_{LU} + \text{sols} \cdot T_{sol}.$$

method (N_{opt})	step	fail	LUs	sols	time
Newton (4)	23	5	86	86	12.2
Newton-chord (7)	23	7	23	125	6.5
ad.Shamanskii (7)	25	9	31	64	5.8

Table 5.2: Results for correctors on the Bratu problem.

The values of T_{LU} and T_{sol} can be estimated from the three results in Table 5.2. If we have $T_{LU} \approx 0.110$ sec and $T_{sol} \approx 0.032$ sec, the total time is approximated very accurate.

For the adaptive Shamanskii method we use equation (4.1), that contains a T_i and T_o, the times for an inner and an outer iteration. An outer iteration contains an LU factorization and the solution of a system with that LU factorization. So we have the relation $T_o = T_{LU} + T_{sol}$ and $T_i = T_{sol}$. The ratio T_i/T_o determines the performance of the adaptive Shamanskii method.

Because of the success of the adaptive Shamanskii method we decided to examine a fourth corrector method, a variant of the adaptive Shamanskii method, that is suggested in [Lev01]. In the first corrector iteration of Newton-chord and adaptive Shamanskii an LU factorization is constructed. This is expensive and maybe we can reuse the LU factorization of the previous step. This results in an adaptive Shamanskii method where the LU factorization can remain fixed over several continuation steps.
The results of the additional test for the fourth corrector are stated in Table 5.3 and can be found in extended form in appendix B.

method (N_{opt})	step	fail	LUs	sols	time
variant of ad.Shamanskii (7)	25	8	25	91	6.4

Table 5.3: Results for variant of ad.Shamanskii on the Bratu problem.

The results for the variant of adaptive Shamanskii are disappointing. They are almost equal to the one of the Newton-chord method. So this variant is at least worse than the normal adaptive Shamanskii method. Another disadvantage is that the total solve time varies between the different initial step sizes more than with the other methods (from 4.8 to 7.7 sec). Because of this unpredictability and because it gives no extra gain in time, we will not consider it as an alternative corrector method anymore.

We remark that the success of the adaptive Shamanskii with respect to the Newton-chord method depends on certain circumstances. The gain in time relies on the properties of the used computer, the problem size and the solver.
The dependence on the computer properties is caused by the adaptive Shamanskii method, that uses the computing times spent on inner and outer iterations. The ratio of those times (T_i/T_o) can change if we switch from computer, due to other processor performance and memory capacity. For the same problem parameters we obtain on one computer a larger factor between the solve time of Newton-chord and adaptive Shamanskii than on another computer.
If the systems is very large, the LU factorization will be very expensive. In that case the adaptive Shamanskii method can hardly gain any time by taking extra Newton steps. However the method will be at least as good as the Newton-chord method.

We have not payed any attention yet to the achievements of the step size control. We illustrate the results here by an example. We picked one of the test runs, the continuation process of the adaptive Shamanskii method ($N_{opt} = 7$) with initial step size 1.0, and plotted it in Figure 5.3. The left figure shows the points on the curve that are found by the algorithm. Near the turning

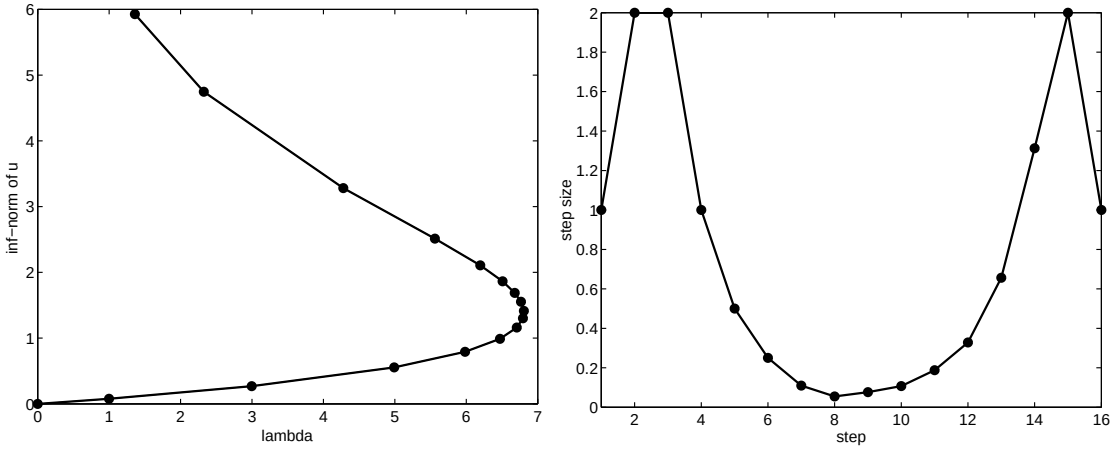

Figure 5.3: Points on the bifurcation diagram of the Bratu problem (left) and variation of step size (right) for adaptive Shamanskii ($N_{opt} = 7$) with initial step size 1.0.

point, the points lie close together, that means the step size is small. On the contrary on the flat branches where points lie far from each other, the step size is large. This can also be concluded form the right figure, where the size of each step is plotted.

We can conclude that the combination of adaption by N_{opt} and trial and error for automatic step size adaption works well. The algorithm never jammed, so we needed no manual restarts. Just before the turning point the corrector fails to converge for a few steps. These steps were repeated automatically with a smaller step size. After the turning point the step size increases to the maximum allowed step size rapidly.

5.4 Results for Ocean Circulation

To test the correctors and step size control for the ocean circulation problem we used almost the same approach as for the Bratu problem. We compare the three correctors with the same choice for N_{opt}. We run the continuation process with a few initial step sizes between 0.3 and 1.1. Finally we take the average over all initial step sizes to compare the correctors. The results are shown in Table 5.4, where we use the abbreviations form Table 5.1. See appendix C for the full test results.

method (N_{opt})	step	fail	ILUs	sols	time
Newton (4)	23	2	94	94	1586
Newton-chord (7)	35	5	35	189	782
Ad.Shamanskii (7)	29	4	38	149	765

Table 5.4: Results for the ocean circulation problem.

In the ocean circulation problem there are more unknowns than in the Bratu problem, so we have to solve a bigger system. This has some effect on the results. Again Newton-chord and adaptive Shamanskii use half the time of the Newton method, but now the difference between both methods in time is very small. We have already explained this in the previous section.

To illustrate the step size control in case of the ocean circulation problem, we add Figure 5.4. The left graph shows the points on the curve obtained by the adaptive Shamanskii ($N_{opt} = 7$) method with initial step size 0.7. The variation of the step size along the branch can be viewed

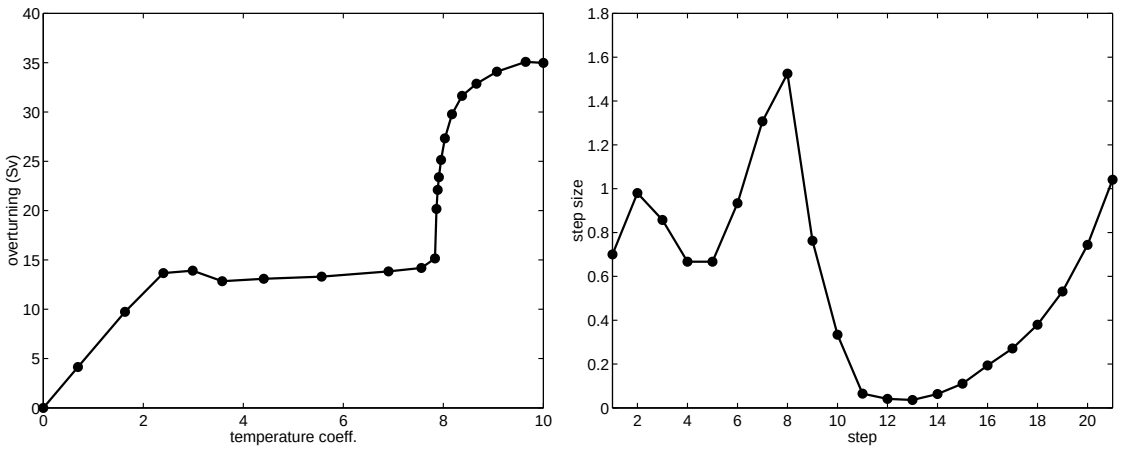

Figure 5.4: Points on the bifurcation diagram of the ocean model (left) and variation of step size (right) for adaptive Shamanskii ($N_{opt} = 7$) with initial step size 0.7.

in the right graph.
We saved the solution in two points on the branch, one before and one after the pitchfork bifurcation. We made some plots of the solution to get some insight in the bifurcation. These can be found in appendix D.

5.5 Conclusions

We are at the end of part I of this thesis, so it is time to draw some conclusions.
We proposed in Chapter 3 a method to choose the step size automatically with use of N_{opt} and trial and error. For the two test problems, Bratu and ocean circulation, the step size control works well. We need no manual restarts and the method can handle difficult points as a turning point and a pitchfork bifurcation where the step size has to be small.
The test for the correctors that we introduced in Chapter 4 is in favor of the adaptive Shamanskii method. It beats the Newton method by far and is as expected a little faster than the Newton-chord method.

Both step size control and the adaptive Shamanskii method have been successfully implemented in THCM 3.1, the latest version of the continuation program for thermohaline ocean circulation. To determine the approximate speed up generated by this improvement, we performed a huge ocean circulation test. We ran a few steps for parameter 27, that controls the vertical diffusivity, for Newton with a fixed step size, for adaptive Shamanskii with fixed step size and finally for adaptive Shamanskii with step size control. The results are stated in Table 5.5.

The values in this table may be a little different for an other part of the branch or another parameter, nevertheless they show that the improvements give an important speed up to the continuation process.

METHOD	TIME
Newton ($N_{opt} = 4$) with fixed step size	27547
ad.Shamanskii ($N_{opt} = 7$) with fixed step size	13080
ad.Shamanskii ($N_{opt} = 7$) with step size control	6418

CHANGE	SPEED-UP
Newton $\Rightarrow$ adaptive Shamanskii	2.1
fixed steps $\Rightarrow$ step size control	2.0
OVERALL	4.3

Table 5.5: Speed-up by the improvements.

Part II

Chapter 6

The Saddle Point Problem

In each continuation step for the ocean circulation problem we have to solve equations with the Jacobian matrix Φ a number of times. To obtain the solution, we construct a preconditioner with MRILU and then we use a GMRES process to solve the equation. The construction of a preconditioner is very expensive for large systems even with a relatively fast algorithm as MRILU. We have already reduced the total time in each continuation step by introducing the adaptive Shamanskii method, that smartly reuses the last constructed preconditioner.
For further improvements of the continuation algorithm we examine the system solver. Maybe there are good alternatives for MRILU that provide a better preconditioner using less time and therefore we studied the literature on solvers for (Navier)-Stokes euqtion.

In this chapter we introduce the basic equations in fluid dynamics, that also play an important role in the thermohaline ocean circulation model. The Navier-Stokes equations and all relevant variants and linearizations can be formulated as a saddle point problem. Finally we will pay some attention to the specific aspects of the equations of ocean circulation.
In the next chapters we will introduce a wide range of solvers for the saddle point problem and try to assess the value of the different approaches. In Chapter 9 we examine a possible improvement of MRILU that can be useful for the saddle point problem.

6.1 Governing Equations

We examine the following important equations in fluid mechanics:

$$-\nu\Delta u + (w\cdot\nabla)u + \nabla p = f \text{ in } \Omega, \tag{6.1}$$

$$\nabla\cdot u = g \text{ in } \Omega, \tag{6.2}$$

$$u = h \text{ on } \partial\Omega, \tag{6.3}$$

where $\Omega \subset \mathbb{R}^2$, ν is the viscosity of the fluid, f represents body forces driving the fluid and h is the prescribed velocity field at the boundary. Often the velocity field is taken divergence free, that means $g = 0$.
We left open the choice of w in equation (6.1). The difficulties to solve the equation largely depends on this choice. We examine three cases.

1. Steady state incompressible Navier-Stokes equation: $w \equiv u$.
2. Stokes equation: $w \equiv 0$.
3. Oseen problem: all other cases with given w independent of u and p.

The *generalized* Stokes equation is obtained if a linear term αu with constant $\alpha > 0$ is added to the left-hand side of equation (6.1). This extra term occurs in unsteady Navier-Stokes calculations

with $\alpha = \delta t^{-1}$, where δt is the size of the time step. In this case the momentum equation becomes

$$-\nu\Delta u + \alpha u + (w \cdot \nabla)u + \nabla p \quad = \quad f. \tag{6.4}$$

6.2 Treatments of the Convective Term

If we want to solve the incompressible Navier-Stokes equations, the convective term $(u\cdot\nabla)u$ causes problems, because it is nonlinear. Usually a linearization is made for this term. There are several ways to do this.

Assume we have U independent of u and p, close to the solution u. If the error $\delta u = U - u$ is small, we have a good approximation for the solution and we can use U to linearize the convective term.
If the Navier Stokes equations are solved by a fixed point iteration, we have $u_{k+1} = u_k + \delta u$. Now we can use the previous solution (u_k) as an approximation for the next one (u_{k+1}) if δu is small. Therefore in the next four approaches for the convective term usually

$$u = u_{k+1} \text{ and } U = u_k.$$

John et al. in [JMM+00] simply use $(U \cdot \nabla)u$ as approximation for $(u \cdot \nabla)u$. The error is of first order in δu. Hence in this case we have to solve the Oseen problem.

Olshanskii uses in [Ols99] the equality

$$(u \cdot \nabla)u = (\text{curl } u) \times u + \nabla \left(\frac{u^2}{2}\right)$$

and replaces (curl u) $\times u$ by (curl U) $\times u$. The gradient term is added to the gradient of the pressure which results in the Bernoulli pressure $P = p + \frac{u^2}{2}$. This linearization has an advantage for the discretization in the next section. Matrix A of equation (6.6) involves the term (curl U) $\times u$, that is zeroth order in u, instead of $(U \cdot \nabla)u$, that is of first order because it contains a derivative of u. In literature this problem with in equation (6.1) a term $(w \times u)$ instead of $(w \cdot \nabla)u$ is also called Oseen problem.

In case of a divergence free velocity field the equality

$$(u \cdot \nabla)u = (\nabla \cdot u\ u) - u(\nabla \cdot u) = (\nabla \cdot u\ u) \tag{6.5}$$

can be used to replace the convective term. The equation does not change much, but according to [Vel94] in problems with large gradients the discretization of the conservation form $(\nabla \cdot u\ u)$ causes less problems than the discretization of $(u \cdot \nabla u)$. In this case we still have to deal with the non-linearity.

Finally we can use *Newton linearization* to deal with the nonlinearity of the convective term. Consider the following equality for the convective term with use of $u = U + \delta u$,

$$(u \cdot \nabla)u = (U \cdot \nabla)u + (u \cdot \nabla)U - (U \cdot \nabla)U + (\delta u \cdot \nabla)\delta u.$$

The last term is $O(\delta u^2)$ and can be dropped if δu is small. So the error is not of first order as in the approach of John and Olshanskii but of second order.

6.3 Definitions of Matrix Properties

Discretization of the equations above gives a matrix. The properties of this matrix determine the best method to solve the discretized equation. In this section we define some properties, that will

be frequently used in the next chapters.

First we define symmetry for a matrix

Definition 6.1. *A real square matrix M is symmetric if $\langle Mx, y\rangle = \langle x, My\rangle$ for all x and y.*

In this definition symmetry is related to the inner product. In case of the standard Euclidean inner product it follows that $M^T = M$. A symmetric matrix has real eigenvalues.

The second important matrix property is definiteness

Definition 6.2. *A real square matrix M is positive definite if $\langle Mx, x\rangle > 0$ for all x.*

Definition 6.3. *A real square matrix M is positive semidefinite if $\langle Mx, x\rangle \geq 0$ for all x.*

Definition 6.4. *A real square matrix M is indefinite if $\langle Mx, x\rangle > 0$ for some x and < 0 for some other x.*

A real matrix is positive definite if and only if all the eigenvalues have a positive real part. If the matrix is also symmetric the eigenvalues are both positive and real. We can split a real matrix M in a symmetric part $(M + M^T)/2$ and a skew-symmetric part $(M - M^T)/2$. The skew-symmetric part has no influence on the definiteness of the matrix. If the symmetric part is positive definite, then M is also positive definite.
We use the symbol '$\succ$' to denote the definiteness of the matrix. So $M \succ 0$ means that the matrix M is positive definite and $M \succeq 0$ that it is positive semi-definite.
Some authors (for example Bramble and Pasciak [BP88]) restrict the definition of a positive definite matrix to symmetric matrices. This can be a little confusing.

In general the product of two symmetric and positive definite matrices is neither symmetric nor positive definite. However if M and N are positive definite then $N^T MN$ is positive definite too. This can be seen directly from $\langle N^T MNx, x\rangle = \langle M(Nx), Nx\rangle \geq 0$.

6.4 Discretization

Commonly used finite element and finite difference discretizations of the Oseen equations lead to the saddle point problem

$$K \begin{pmatrix} u \\ p \end{pmatrix} \equiv \begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}. \tag{6.6}$$

In this equation the matrix B is the discrete divergence operator. B is the adjoint of B^T, the discrete gradient operator. The properties of A depend on the choice of w and the boundary condition. In case of the Stokes problem with Dirichlet boundary A is symmetric and positive definite. Hence K is symmetric and has real eigenvalues. Since

$$\langle Kx, x\rangle = (u^T, p^T) \begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = u^T Au + 2u^T B^T p,$$

for any u_* such that $Bu_* \neq 0$, there exists a p_* that makes this product postive and another p_* that makes it negative. So the problem is symmetric and indefinite.
Apart from (6.6) we can choose another formulation of the problem. Instead of equation (6.2) we can use its opposite $(-\nabla \cdot u = -g)$. If we do this $-B$ appears in the lower left part of the matrix and it belongs with B^T in the upper right to the skew-symmetric part of the matrix. In this case we have

$$\langle \bar{K}x, x\rangle = (u^T, p^T) \begin{pmatrix} A & B^T \\ -B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = u^T Au.$$

For any vector $x = (0, p_*)$ this product equals zero, hence K is semidefinite.

In the case of the Stokes equation with Dirichlet boundary conditions, the matrix A is the direct sum of two or three (depending on the dimension of the problem) matrices representing discretized Laplace operators. For the related discretized Poisson problems very fast solving techniques are available.

6.5 Remarks on Ocean Circulation

In the three-dimensional thermohaline ocean model we have velocities in three directions, so we can write $u = (u_1, u_2, u_3)$. The equations differ a little from the Navier-Stokes equation. Most changes are caused by the choice of a spherical coordinate system rotating with the earth. For an extended description see [DOWB01]. Important is the addition of an extra term caused by the coriolis forces to equation (6.1). This term is $(-\Omega u_2, \Omega u_1, 0)$, where Ω is a variable related to the angular velocity of the rotating sphere and the degree of latitude. So we get cross terms between velocity in longitudinal and latitudinal direction.
The main disadvantage is that we get after discretization a non symmetric component in the matrix A, caused by the cross terms. Because the extra term is skew-symmetric, A is still positive definite. However the saddle point matrix K is now as well non-symmetric as indefinite.

In the thermohaline ocean model next to velocity and pressure two other variables play an important role: salinity and temperature. In the next chapters we will forget these variables for a moment. We concentrate on approaches for saddle point problems with a symmetric positive definite matrix A. The next step will be an application to non-symmetric A. We assume that the major troubles in solving the discretized momentum equation are caused by the asymmetry of the problem. Generalization of a method to the ocean circulation model with the equations of salinity and temperature can be done later.

Chapter 7

Basic Iterative Methods

If we deal with large sparse systems the saddle point problem can be solved best with an iterative method. There is a wide range of iterative methods available. In this chapter we introduce some of them. We restrict ourselves to the relevant solving techniques for equation (6.6).
In general we can split the solvers in two classes. The solvers in the first class are the black-box solvers, that don't use much information about a problem. They can be applied directly to the saddle point problem. A short introduction to this type is given in Section 7.1.
The solvers in the other class try to exploit the properties of the saddle point equation. The iterative methods in this class make use of several alternative formulations of the saddle point problem. In sections 7.2-7.9 we discuss a number of possible approaches. We follow partially the exposition of Zulehner, who has done an extended research on two reformulations in [Zul02].
We discuss *basic* iterative methods, that means we stop after some splitting, reformulation or stabilization of the system. We replace one system by another with new, better properties. However we still need to solve that new system with an iterative method or preconditioner. The actual solvers are discussed in the next chapter. Here we stick at the formulation.
Unless explicitly mentioned, we assume in this chapter that we have to deal with a symmetric positive definite saddle point problem.

7.1 Block Approach

A simple way of dealing with the saddle point problem is the block approach. First rearrange the unknowns such that velocity and pressure in each cell are grouped. Then generalize a method for scalar equations to a preconditioner for this block-structured matrix. For several methods (ILU factorizations, multigrid) such a generalization is possible.
In an attempt to produce a preconditioner for K without usage of the structure of (6.6) it seems logical to group unknowns that belong together. Velocity and pressure should be eliminated together or not be eliminated at all. For this method appropriate scaling of variables is important. The unknowns and the entries of the matrix should be of the same order of magnitude.
The logical clustering of unknowns is immediately the main disadvantage of this approach. Velocity and pressure are treated in the same way, whereas a separated treatment like in the next sections could be more appropriate.

7.2 Schur-Complement Method

We can split the solution of velocity and pressure in the saddle point problem by computing the Schur-complement. Then we have to solve first

$$Cp \equiv (BA^{-1}B^T)p = BA^{-1}f - g. \tag{7.1}$$

To solve this equation we have to take two steps. Solve $Az = f$ and then solve $Cp = Bz - g$. If we have computed the vector of pressure values we can compute the velocity vector by

$$u = A^{-1}(f - B^T p). \tag{7.2}$$

The disadvantage of this separation is that in general the computation of A^{-1} is expensive. If we use a preconditioner $\hat{A}$ for A to solve equation (7.1) we have to use a nested iteration. An inner iteration to evaluate A^{-1} and an outer iteration to compute C^{-1}. Unfortunately sometimes the inner iteration has to reach a high level of accuracy in order to ensure convergence of the outer iteration.

7.3 Inexact Uzawa Algorithms

To get a better solver an iterative process should solve equation (7.1) and (7.2) simultaneously. Therefore consider the following class of methods

$$\begin{aligned} \hat{A}(u_{k+1} - u_k) &= f - Au_k - B^T p_k, \\ \hat{C}(p_{k+1} - p_k) &= Bu_{k+1} - g, \end{aligned}$$

with $\hat{A}$ and $\hat{C}$ positive definite.
This class of methods is based on the Uzawa algorithm that appears with $\hat{A} = A$ and $\hat{C} = \gamma I$. We get the method of Bramble and Pasciak if we take $\hat{C} = I$.

If we use *artificial compressibility* for the solution of equation (6.2) we get an inexact Uzawa algorithm. Incompressibility is replaced by the artifical gas model $p = c^2\rho$ which leads to the following continuity equation

$$\frac{1}{c^2}\frac{\partial p}{\partial t} + \nabla \cdot u = g. \tag{7.3}$$

After discretization we get an inexact Uzawa algorithm with $\hat{C} = \frac{1}{c^2 \delta t} I$.

The inexact Uzawa method can be seen as a preconditioned Richardson method

$$\hat{K}_1 \begin{pmatrix} u_{k+1} - u_k \\ p_{k+1} - p_k \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} - K \begin{pmatrix} u_k \\ p_k \end{pmatrix},$$

with preconditioner

$$\hat{K}_1 = \begin{pmatrix} \hat{A} & 0 \\ B & -\hat{C} \end{pmatrix}.$$

Let (u_*, p_*) be the exact solution of equation (6.6). For the error

$$\Delta u_k = u_k - u_*, \quad \Delta p_k = p_k - p_*,$$

we have

$$\begin{pmatrix} \Delta u_{k+1} \\ \Delta p_{k+1} \end{pmatrix} = M_1 \begin{pmatrix} \Delta u_k \\ \Delta p_k \end{pmatrix},$$

where M_1 denotes the iteration matrix given by $M_1 = I - \hat{K}_1^{-1} K$. This matrix can be written as a product of two or three symmetric matrices

$$\begin{aligned} M_1 &= -\begin{pmatrix} \hat{A}^{-1} & \hat{A}^{-1}B^T\hat{C}^{-1} \\ \hat{C}^{-1}B\hat{A}^{-1} & \hat{C}^{-1}B\hat{A}^{-1}B^T\hat{C}^{-1} - \hat{C}^{-1} \end{pmatrix} \begin{pmatrix} A - \hat{A} & 0 \\ 0 & \hat{C} \end{pmatrix} && (7.4) \\ &= -\begin{pmatrix} \hat{A}^{-1} & 0 \\ 0 & \hat{C}^{-1} \end{pmatrix} \begin{pmatrix} \hat{A} & B^T \\ B & B\hat{A}^{-1}B^T - \hat{C} \end{pmatrix} \begin{pmatrix} \hat{A}^{-1}(A - \hat{A}) & 0 \\ 0 & I \end{pmatrix} && (7.5) \\ &= P_1 N_1 Q_1. \end{aligned}$$

We remark that the apparently different preconditioner

$$\hat{K}_1^T = \begin{pmatrix} \hat{A} & B^T \\ 0 & -\hat{C} \end{pmatrix},$$

can be reduced to the inexact Uzawa algorithms, if we start with (u_0, p_1) instead of (u_0, p_0). Here p_1 can be computed from $\hat{C}(p_1 - p_0) = Bu_0 - g$. Hence the only difference is the starting point.

7.4 A Class of Symmetric Preconditioners

The inexact Uzawa algorithms provide a *non-symmetric* preconditioner for the *symmetric* saddle point problem. If we prefer a symmetric preconditioner there is an alternative.
Consider the factorization

$$K = \begin{pmatrix} A & 0 \\ B & I \end{pmatrix} \begin{pmatrix} A^{-1} & 0 \\ 0 & -C \end{pmatrix} \begin{pmatrix} A & B^T \\ 0 & I \end{pmatrix},$$

which motivates the following symmetric preconditioner

$$\hat{K}_2 = \begin{pmatrix} \hat{A} & 0 \\ B & I \end{pmatrix} \begin{pmatrix} \hat{A}^{-1} & 0 \\ 0 & -\hat{C} \end{pmatrix} \begin{pmatrix} \hat{A} & B^T \\ 0 & I \end{pmatrix}.$$

Usage of this preconditioner is equivalent to the iterative procedure

$$\begin{aligned} \hat{A}(\hat{u}_{k+1} - u_k) &= f - Au_k - B^T p_k, \\ \hat{C}(p_{k+1} - p_k) &= B\hat{u}_{k+1} - g, \\ \hat{A}(u_{k+1} - \hat{u}_{k+1}) &= -B^T(p_{k+1} - p_k). \end{aligned}$$

This method can be seen as an inexact Uzawa method with one extra step. In this step u is corrected for the change in p.
The iteration matrix $M_2 = I - \hat{K}_2^{-1}K = \hat{K}_2^{-1}(\hat{K}_2 - K) = -\hat{K}_2^{-1}(K - \hat{K}_2)$ now becomes

$$\begin{aligned} M_2 &= -\begin{pmatrix} \hat{A} & B^T \\ B & B\hat{A}^{-1}B^T - \hat{C} \end{pmatrix}^{-1} \begin{pmatrix} A - \hat{A} & 0 \\ 0 & \hat{C} - B\hat{A}^{-1}B^T \end{pmatrix} \\ &= P_2 N_2 Q_2. \end{aligned} \tag{7.6}$$

Here we have $P_2 = \pm I$, so again the iteration matrix can be written as a product of two symmetric matrices.

7.5 Reformulation

In both the inexact Uzawa methods and the symmetric preconditioners the iteration matrix can be generally written as

$$M = PNQ,$$

with block diagonal matrices P, Q and symmetric matrix N.
Zulehner now introduces two block diagonal matrices D and E, that are symmetric and positive definite. Then consider the following transformation

$$\bar{M} = D^{1/2} M D^{-1/2} = \left[D^{1/2} P E^{1/2}\right] \left[E^{-1/2} N E^{-1/2}\right] \left[E^{1/2} Q D^{-1/2}\right] = \bar{P}\bar{N}\bar{Q}. \tag{7.7}$$

First case: Assume D and E can be chosen such that

$$\bar{P} = \bar{Q} = I \tag{7.8}$$

It follows immediately that $\bar{M} = \bar{N}$ which implies

1. M is symmetric with respect to the scalar product $\langle ., . \rangle_D$ given by

$$\langle x, y \rangle_D = \langle Dx, y \rangle$$

where $\langle ., . \rangle$ denotes the ordinary Euclidean scalar product;

2. the spectra of M and $\bar{N}$ coincide:

$$\sigma(M) = \sigma(\bar{N})$$

3. the convergence rate of the iteration method is given by

$$\rho(M) = \|M\|_D = \rho(\bar{N})$$

Here, $\rho(.)$ denotes the spectral radius and $\|.\|_D$ the norm associated to the scalar product $\langle ., . \rangle_D$.

Equation (7.8) can be satisfied if $P = I$ and Q is symmetric and positive definite. Then the transformation matrices are given by

$$D = Q, \quad \text{and} \quad E = Q^{-1}. \tag{7.9}$$

Second case: There are other possibilities beside the choice of (7.8). For example assume that D and E can be chosen such that

$$\bar{P} = -\bar{Q} = I.$$

Then $\bar{M} = -\bar{N}$ with similar implications as before. This case occurs if $P = I$ and Q symmetric and negative definite. Then the transformation matrices become

$$D = -Q, \quad \text{and} \quad E = -Q^{-1}. \tag{7.10}$$

General case: If it is not possible to reach one of the strong results above for $\bar{P}$ and $\bar{Q}$ there is another possibility. If there exist transformation matrices E and D with

$$\|\bar{P}\| \leq 1 \text{ and } \|\bar{Q}\| \leq 1, \tag{7.11}$$

then we still can conclude

$$\rho(M) = \|M\|_D \leq \rho(\bar{N}).$$

Necessary condition for E is

$$QPEP^T Q^T \preceq E^{-1}.$$

In all relevant cases QP is symmetric and then this condition is equivalent to $(E^{1/2} QPE^{1/2})^2 \preceq I$. If this equation is solved, then D is obtained from

$$D = (PEP^T)^{-1}.$$

7.6 Convergence Analysis for Inexact Uzawa Algorithms

In this section we examine the convergence properties of the inexact Uzawa algorithms after reformulation. For the analysis two different cases are distinguished.
Assume the following inequalities hold for the preconditioners $\hat{A}$ and $\hat{C}$

$$\alpha_1 \hat{A} \preceq A \preceq \alpha_2 \hat{A} \tag{7.12}$$

with constants $\alpha_1, \alpha_2 \in \mathbb{R}$ such that $0 < \alpha_1 \leq 1 \leq \alpha_2$, and

$$\gamma_1 \hat{C} \preceq BA^{-1}B^T \preceq \gamma_2 \hat{C} \tag{7.13}$$

with constants $\gamma_1, \gamma_2 \in \mathbb{R}$ such that $0 < \gamma_1 \leq \gamma_2$.

Special case: This case corresponds to the first case in the previous section, so we demand $\bar{P} = \bar{Q} = I$. The transformation is based on equation (7.4), where we have $P = I$ and Q symmetric. For the reformulation Q has to be positive definite too. So $A - \hat{A}$ has to be positive definite and therefore we take $\alpha_1 = 1$ in equation (7.12). Now $D = Q$ and $E = Q^{-1}$, which defines the following scalar product

$$\left\langle \begin{pmatrix} u \\ p \end{pmatrix}, \begin{pmatrix} v \\ q \end{pmatrix} \right\rangle_D = \langle (A - \hat{A})u, v \rangle + \langle \hat{C}p, q \rangle. \tag{7.14}$$

This scalar product is used in the following theorem.

Theorem 7.1. *Let $A, \hat{A}$ be symmetric, positive definite $n \times n$ matrices, B a $m \times n$ matrix, $\hat{C}$ a symmetric, positive definite $m \times m$ matrix, satisfying* (7.12) *with* $\alpha_1 = 1$ *and* (7.13).

Then we have:

1. *The iteration matrix M_1, of the inexact Uzawa algorithm is symmetric with respect to the scalar product* (7.14).
2. $\sigma(\bar{N}_1) \subset [\rho_1, \rho_2] \subset (-\infty, 1)$ *with*

$$\begin{aligned} \rho_1 &= \frac{2 - (1+\gamma_2)\alpha_2}{2} - \sqrt{\frac{[2 - (1+\gamma_2)\alpha_2]^2}{4} + \alpha_2 - 1}, \\ \rho_2 &= \frac{2 - (1+\gamma_1)\alpha_2}{2} + \sqrt{\frac{[2 - (1+\gamma_1)\alpha_2]^2}{4} + \alpha_2 - 1}. \end{aligned}$$

3. *If* $\alpha_2(2 + \gamma_2) < 4$, *then* $\rho(M_1) = \|M_1\|_D < 1$.
4. $\hat{K}_1^{-1}K$ *is symmetric and positive definite with respect to the scalar product* (7.14) *and* $\sigma(\hat{K}_1^{-1}K)) \subset [1 - \rho_2, 1 - \rho_1] \subset (0, \infty)$.

Proof. See [Zul02]. □

General case: In the general case a symmetric and positive definite reformulation is not possible. However we can define another scalar product that provides us sharp convergence estimates. We have to chose matrices D and E such that (7.11) is fulfilled. We introduce the following constants

$$\alpha_0 = \frac{\alpha_1 + \alpha_2 - 2}{\alpha_2 - \alpha_1} \quad \text{and} \quad \hat{\alpha} = \frac{\alpha_1 + \alpha_2 - 2\alpha_1\alpha_2}{\alpha_2 - \alpha_1},$$

that are used in the matrix D (we do not need to know E explicitely)

$$D = \begin{pmatrix} \hat{\alpha}\hat{A} + \alpha_0 A & 0 \\ 0 & \hat{C} \end{pmatrix}.$$

This matrix induces the scalar product

$$\left\langle \begin{pmatrix} u \\ p \end{pmatrix}, \begin{pmatrix} v \\ q \end{pmatrix} \right\rangle_D = \langle (\hat{\alpha}\hat{A} + \alpha_0 A)u, v \rangle + \langle \hat{C}p, q \rangle. \tag{7.15}$$

Now the following theorem holds for the inexact Uzawa algorithms in the general case. In this theorem $\bar{N}_1 = E^{-1/2}N_1E^{-1/2}$ with N_1 as in equation (7.5).

Theorem 7.2. *Let $A, \hat{A}$ be symmetric, positive definite $n \times n$ matrices, B a $m \times n$ matrix, $\hat{C}$ a symmetric, positive definite $m \times m$ matrix, satisfying* (7.12) *and* (7.13).

Then we have with $\|\cdot\|_D$ the norm induced by (7.15)*:*

1. $\sigma(\bar{N}_1) \subset [\min(-q_\alpha, \rho_1, \rho_3), \max(\rho_2, \rho_4)]$ *with*

$$\begin{aligned}
q_\alpha &= \max(1-\alpha_1, \alpha_2 - 1), \\
\rho_1 &= \frac{2-(1+\gamma_2)\alpha_2}{2} - \sqrt{\frac{[2-(1+\gamma_2)\alpha_2]^2}{4} + \alpha_2 - 1}, \\
\rho_2 &= \frac{2-(1+\gamma_1)\alpha_2}{2} + \sqrt{\frac{[2-(1+\gamma_1)\alpha_2]^2}{4} + \alpha_2 - 1}, \\
\rho_3 &= \frac{(1-\gamma_2)\alpha_1}{2} - \sqrt{\frac{(1-\gamma_2)^2\alpha_1^2}{4} + 1 - \alpha_1}, \\
\rho_4 &= \frac{(1-\gamma_1)\alpha_1}{2} + \sqrt{\frac{(1-\gamma_1)^2\alpha_1^2}{4} + 1 - \alpha_1}.
\end{aligned}$$

2. $\|M_1\|_D \leq \max(q_\alpha, -\rho_1, -\rho_3, \rho_2, \rho_4)$.

3. *If* $\alpha_2(2+\gamma_2) < 4$, *then* $\|M_1\|_D < 1$.

Proof. See [Zul02] □

This theorem provides sharp estimates for the quality of the inexact Uzawa algorithms.

In many cases a simple preconditioner for C is chosen: a diagonal matrix, $\hat{C} = \Gamma$, for example the pressure mass matrix, or even simplier $\hat{C} = \gamma I$. The choice for the last preconditioner limits the convergence process. This is stated in the following theorem.

Theorem 7.3. *Let $A, \hat{A}$ be symmetric, positive definite $n \times n$ matrices, B a $m \times n$ matrix, $\hat{C} = \gamma I$ with γ a positive real number and I the $m \times m$ identity matrix, satisfying* (7.12) *and* (7.13). *Let γ_{opt} denote the optimal value for γ. Further assume $\sigma(C) \subset [\mu_{min}, \mu_{max}]$.*

Then we have:

1. $\gamma_{opt} = (\mu_{max} + \mu_{min})/2$.

2. *If $\gamma = \gamma_{opt}$, then $\kappa(\hat{K}_1^{-1}K) \leq \kappa(C)$.*

Proof. Note that theorem 7.2 can be applied, because $\hat{C} = \gamma I$ is symmetric and positive definite. Suppose we require $\|M_1\|_D \leq c$ with constant $0 \leq c \leq 1$. Then according to statement 2 of the previous theorem $q_\alpha, -\rho_1, -\rho_3, \rho_2$ and ρ_4 all have to be smaller than c. For q_α this leads straight to the inequality

$$1 - c \leq \alpha_1 \leq 1 \leq \alpha_2 \leq 1 + c. \tag{7.16}$$

For the other constants the calculations are more complicated. Define $k = 1 - (1+\gamma_2)\alpha_2/2$ then the estimate for ρ_1 becomes

$$-\rho_1 \leq c \;\Rightarrow\; -k + \sqrt{k^2 + \alpha_2 - 1} \leq c \;\Rightarrow\; \alpha_2 - 1 \leq 2ck + c^2.$$

If k is substituted, we can bring γ_2 to the left side. With use of (7.16) we get the following inequality

$$\gamma_2 \leq \frac{1}{c}\left[\frac{(c+1)^2}{\alpha_2} - (c+1)\right] \leq 1 + c.$$

This result is also obtained if we evaluate $-\rho_3 \leq c$.
For ρ_2 we use a similar procedure. If we define $m = (1-\gamma_2)\alpha_1/2$, then

$$\rho_2 \leq c \;\Rightarrow\; m + \sqrt{m^2 + 1 - \alpha_1} \leq c \;\Rightarrow\; 1 - \alpha_1 \leq -2cm + c^2.$$

If m is substituted, then γ_1 can be brought to the left side. After some basic calculations and with use of (7.16) we get

$$\gamma_1 \geq \frac{1}{c}\left[(1-c) - \frac{(1-c)^2}{\alpha_2}\right] \geq 1 - c.$$

Evaluation of $\rho_4 \leq c$ leads to the same inequality.
Combination of the estimates for γ_1 and γ_2 gives

$$1 - c \leq \gamma_1 \leq \gamma_2 \leq 1 + c. \tag{7.17}$$

Summarizing, if we want $\|M_1\| \leq c$, the preconditioners $\hat{A}$ and $\hat{C} = \gamma I$ have to fulfill (7.16) and (7.17). Assume that for arbitrary values of c we can build $\hat{A}$ such that (7.16) is fulfilled. Now we have to choose γ. We use (7.13) in combination with an inequality that follows from the spectrum of C and its positive definiteness,

$$\mu_{min} I \preceq B A^{-1} B^T \preceq \mu_{max} I. \tag{7.18}$$

Now c is as small as possible if we let the bounds of (7.13) and (7.18) coincide. That gives

$$\mu_{min} = (1 - c)\gamma \quad \text{and} \quad \mu_{max} = (1 + c)\gamma.$$

In this system γ and c can be solved and expressed as functions of the maximal and minimal eigenvalue or even the condition number $\kappa(C)$. The solution is

$$\gamma = \frac{\mu_{max} + \mu_{min}}{2} \quad \text{and} \quad c = \frac{\mu_{max} - \mu_{min}}{\mu_{max} + \mu_{min}} = \frac{\kappa(C) - 1}{\kappa(C) + 1}.$$

We have proven the first statement. The second statement can be derived immediately. If the triangle inequality is applied to $\hat{K}_1^{-1} K = I - M_1$ we get

$$(1 - c) \leq (1 - \|M_1\|_D) \leq \|\hat{K}_1^{-1} K\|_D \leq (1 + \|M_1\|_D) \leq (1 + c). \tag{7.19}$$

With this inequalities we can estimate the condition number by

$$\kappa(\hat{K}_1^{-1} K) \leq \frac{1 + c}{1 - c} = \kappa(C).$$

□

This theorem holds also in the special case, where we have $\alpha_1 = 1$.
We can conclude that even with optimal choice for γ convergence in the inexact Uzawa method is limited by the condition number of C. If $\kappa(C)$ is large a better choice for $\hat{A}$ with α_1 and α_2 closer to one will not improve the method.
The theorem provides an optimal value γ, that can be transformed to an optimal value for the constant in the artificial gas model, used in artificial compressibility, equation (7.3). In case of the Stokes equation sharp estimates for the eigenvalues of C are available.
Elman obtained in [Elm01] the same estimate for γ_{opt}.

7.7 Convergence Analysis for Symmetric Preconditioners

In this section we examine the convergence properties of the symmetric preconditioners after reformulation. The convergence analysis is split in three different cases, which correspond to the cases in Section 7.5. We use the formulation of equation (7.6), where we have $P = I$ and Q symmetric.
Assume the following inequalities hold for the preconditioners $\hat{A}$ and $\hat{C}$

$$\alpha_1 \hat{A} \preceq A \preceq \alpha_2 \hat{A} \tag{7.20}$$

with constants $\alpha_1, \alpha_2 \in \mathbb{R}$ such that $0 < \alpha_1 \leq 1 \leq \alpha_2$, and

$$\beta_1 \hat{C} \preceq B \hat{A}^{-1} B^T \preceq \beta_2 \hat{C} \tag{7.21}$$

with constants $\beta_1, \beta_2 \in \mathbb{R}$ such that $0 < \beta_1 \leq 1 \leq \beta_2$. Note the difference with equation (7.13).

First case: We want $\bar{P} = \bar{Q} = I$, which is possible with positive definite Q. Then we have the transformation matrices $D = Q$ and $E = Q^{-1}$. The matrix Q is positive definite if and only if the components $A - \hat{A}$ and $\hat{C} - B\hat{A}^{-1}B^T$ are positive definite. Therefore we choose $\alpha_1 = 1$ in equation (7.20) and $\beta_2 = 1$ in equation (7.21).
The scalar product induced by D becomes

$$\left\langle \begin{pmatrix} u \\ p \end{pmatrix}, \begin{pmatrix} v \\ q \end{pmatrix} \right\rangle_D = \langle (A - \hat{A})u, v \rangle + \langle (\hat{C} - B\hat{A}^{-1}B^T)p, q \rangle. \tag{7.22}$$

This scalar product is used in the following theorem.

Theorem 7.4. *Let $A, \hat{A}$ be symmetric, positive definite $n \times n$ matrices, B a $m \times n$ matrix, $\hat{C}$ a symmetric, positive definite $m \times m$ matrix, satisfying* (7.20) *with* $\alpha_1 = 1$ *and* (7.21) *with* $\beta_2 = 1$.
Then we have:

1. *The iteration matrix M_2, of the iteration method with symmetric preconditioner is symmetric with respect to the scalar product* (7.22).
2. $\sigma(M_2) \subset [1 - \alpha_2, \rho_2] \subset (-\infty, 1)$ *with*

$$\rho_2 = \frac{(2 - \alpha_2)(1 - \beta_1)}{2} + \sqrt{\frac{(2 - \alpha_2)^2(1 - \beta_1)^2}{4} + (\alpha_2 - 1)(1 - \beta_1)}.$$

3. *If* $\alpha_2 < 2$, *then* $\rho(M_2) = \|M_2\|_D \leq 1$.
4. $\hat{K}_2^{-1}K$ *is symmetric and positive definite with respect to the scalar product* (7.22) *and* $\sigma(\hat{K}_2^{-1}K) \subset [1 - \rho_2, \alpha_2] \subset (0, \infty)$.

Proof. See [Zul02]. □

Second case: We want $\bar{P} = -\bar{Q} = I$, that is possible with negative definite Q. Then we have the transformation matrices $D = -Q$ and $E = -Q^{-1}$. The matrix Q is negative definite if and only if the components $A - \hat{A}$ and $\hat{C} - B\hat{A}^{-1}B^T$ are negative definite. Therefore we choose $\alpha_2 = 1$ in equation (7.20) and $\beta_1 = 1$ in equation (7.21).
The scalar product induced by D becomes

$$\left\langle \begin{pmatrix} u \\ p \end{pmatrix}, \begin{pmatrix} v \\ q \end{pmatrix} \right\rangle_D = \langle (\hat{A} - A)u, v \rangle + \langle (B\hat{A}^{-1}B^T - \hat{C})p, q \rangle. \tag{7.23}$$

With this scalar product we have a similar theorem as in the first case.

Theorem 7.5. *Let $A, \hat{A}$ be symmetric, positive definite $n \times n$ matrices, B a $m \times n$ matrix, $\hat{C}$ a symmetric, positive definite $m \times m$ matrix, satisfying* (7.20) *with* $\alpha_2 = 1$ *and* (7.21) *with* $\beta_1 = 1$.
Then we have:

1. *The iteration matrix M_2, of the iteration method with symmetric preconditioner is symmetric with respect to the scalar product* (7.22).
2. $\sigma(M_2) \subset [-\rho_1, 1 - \alpha_1] \subset (-\infty, 1)$ *with*

$$\rho_1 = \frac{(2 - \alpha_1)(\beta_2 - 1)}{2} + \sqrt{\frac{(2 - \alpha_1)^2(\beta_2 - 1)^2}{4} + (1 - \alpha_1)(\beta_2 - 1)}.$$

3. *If* $\beta_2 < 1 + 1/(3 - 2\alpha_1)$, *then* $\rho(M_2) = \|M_2\|_D \leq 1$.
4. $\hat{K}_2^{-1}K$ *is symmetric and positive definite with respect to the scalar product* (7.23) *and* $\sigma(\hat{K}_2^{-1}K) \subset [\alpha_1, 1 + \rho_1] \subset (0, \infty)$.

Proof. See [Zul02]. □

General case: In the previous cases, it was possible to construct a scalar product that makes the preconditioned system symmetric and positive definite. Unfortunately for the general case this is not possible. However there exist a scalar product that provides sharp convergence estimates. We want $\|\bar{P}\| \leq 1$ and $\|\bar{Q}\| \leq 1$, as in equation (7.11). Consider the following constants

$$\beta_0 = \frac{\beta_1 + \beta_2 - 2}{\beta_2 - \beta_1}, \quad \text{and} \quad \hat{\beta} = \frac{\beta_1 + \beta_2 - 2\beta_1\beta_2}{\beta_2 - \beta_1},$$

that are used to construct the matrix D (we do not need to know E explicitely),

$$D = \begin{pmatrix} q_\alpha \hat{A} & 0 \\ 0 & \hat{\beta}\hat{C} + \beta_0 B\hat{A}^{-1}B^T \end{pmatrix},$$

with $q_\alpha = \max(1-\alpha_1, \alpha_2 - 1)$. This matrix induces the scalar product

$$\left\langle \begin{pmatrix} u \\ p \end{pmatrix}, \begin{pmatrix} v \\ q \end{pmatrix} \right\rangle_D = q_\alpha \langle \hat{A}u, v\rangle + \langle (\hat{\beta}\hat{C} + \beta_0 B\hat{A}^{-1}B^T)p, q\rangle. \tag{7.24}$$

Now the following theorem holds for the symmetric preconditioners in the general case. In this theorem $\bar{N}_2 = E^{-1/2}N_2E^{-1/2}$ with N_2 as in equation (7.6).

Theorem 7.6. *Let $A, \hat{A}$ be symmetric, positive definite $n \times n$ matrices, B a $m \times n$ matrix, $\hat{C}$ a symmetric, positive definite $m \times m$ matrix, satisfying* (7.20) *and* (7.21).

Then we have with $\|\cdot\|_D$ the norm induced by (7.24)*:*

1. *$\sigma(\bar{N}_2) \subset [-q_\alpha, \max(\rho_1, \rho_2)]$ with*

$$\begin{aligned} q_\alpha &= \max(1-\alpha_1, \alpha_2 - 1), \\ \rho_1 &= \frac{(1-q_\alpha)(1-\beta_1)}{2} + \sqrt{\frac{(1-q_\alpha)^2(1-\beta_1)^2}{4} + q_\alpha(1-\beta_1)}, \\ \rho_2 &= \frac{(1+q_\alpha)(\beta_2-1)}{2} + \sqrt{\frac{(1+q_\alpha)^2(\beta_2-1)^2}{4} + q_\alpha(\beta_2-1)}. \end{aligned}$$

2. *$\|M_2\|_D \leq \max(q_\alpha, \rho_1, \rho_2)$.*

3. *If $\alpha_2 < 2$ and $\beta_2 < 1 + 1/(1+2q_\alpha)$, then $\|M_2\|_D < 1$.*

Proof. See [Zul02]. □

This theorem provides sharp estimates for the quality of the symmetric preconditioners.

For the symmetric preconditioner we also examine the simple preconditioner $\hat{C} = \beta I$. We have a similar theorem as for the inexact Uzawa algorithms.

Theorem 7.7. *Let $A, \hat{A}$ be symmetric, positive definite $n \times n$ matrices, B a $m \times n$ matrix, $\hat{C} = \beta I$ with β a positive real number and I the $m \times m$ identity matrix, satisfying* (7.20) *and* (7.21). *Let β_{opt} denote the optimal value for β. Further assume $\sigma(B\hat{A}^{-1}B^T) \subset [\nu_{min}, \nu_{max}]$.*

Then we have:

1. *$\beta_{opt} = (\nu_{max} - \nu_{min})/2$.*

2. *If $\beta = \beta_{opt}$, then $\kappa(\hat{K}_2^{-1}K) \leq \kappa(B\hat{A}^{-1}B^T)$.*

Proof. Because $\hat{C} = \beta I$ is symmetric and positive definite, theorem 7.6 can be applied. Suppose we require $\|M_2\|_D \leq c$ with $0 \leq c \leq 1$. Then according to the second statement in theorem 7.6 q_α, rho_1 and ρ_2 all have to be smaller than c. $q_\alpha < c$ leads directly to the following inequality,

$$1 - c \leq \alpha_1 \leq 1 \leq \alpha_2 \leq 1 + c. \tag{7.25}$$

If we use $k = (1 - q_\alpha)(1 - \beta_1)/2$, the inequality for ρ_1 becomes

$$\rho_1 < c \Rightarrow k + \sqrt{k^2 + q_\alpha(1 - \beta_1)} < c \Rightarrow q_\alpha(1 - \beta_1) < c^2 - 2ck.$$

Now k can be filled in and after some basic calculations using $q_\alpha > 0$, we get

$$\beta_1 > \left[1 - \frac{c^2}{q_\alpha(1 - c) + c}\right] > 1 - c.$$

For ρ_2 the same procedure is followed. If $m = (1 + q_\alpha)(\beta_2 - 1)/2$, then we have

$$\rho_2 < c \Rightarrow m + \sqrt{m^2 + q_\alpha(\beta_2 - 1)} < c \Rightarrow q_\alpha(\beta_2 - 1) < c^2 - 2mc.$$

Substituting m, some reordering and basic calculations using $q_\alpha > 0$ lead to

$$\beta_2 < \left[1 + \frac{c^2}{q_\alpha(1 + c) + c}\right] < 1 + c.$$

If we combine the estimates for β_1 and β_2, we get

$$1 - c < \beta_1 < 1 < \beta_2 < 1 + c \tag{7.26}$$

From the spectrum of $B\hat{A}^{-1}B^T$ we have

$$\nu_{min} I \preceq B\hat{A}^{-1}B^T \preceq \nu_{max} I.$$

The fastest convergence, that is the smallest c is obtained if this bounds coincide with the one in equation (7.21). That means

$$\nu_{min} = \beta(1 - c) \quad \text{and} \quad \nu_{max} = \beta(1 + c).$$

This system can be solved for β and c. The solution is

$$\beta = \frac{\nu_{max} + \nu_{min}}{2} \quad \text{and} \quad c = \frac{\nu_{max} - \nu_{min}}{\nu_{max} + \nu_{min}} = \frac{\kappa - 1}{\kappa + 1},$$

with $\kappa = \kappa(B\hat{A}^{-1}B^T)$. This value for β is β_{opt}, so the first statement of the theorem is proven. The second statement follows immediately if we use equation (7.19) with M_2 and K_2 instead of M_1 and K_1, then

$$\kappa(\hat{K}_2^{-1}K) \leq \frac{1 + c}{1 - c} = \kappa.$$

□

If Theorem 7.7 is compared with Theorem 7.3 one can see that in case of the primitive preconditioner $\hat{C} = \gamma I$, the estimates for the condition numbers of the class of symmetric preconditioners and the inexact Uzawa algorithms are the same. However the symmetric preconditioners require the solution of an extra equation with $\hat{A}$ in each iteration step. In this case the costs of this extra step are not compensated by a better convergence so the symmetric preconditioners are expected to be more expensive than the inexact Uzawa algorithm.

7.8 Remarks on Reformulation

We make a few remarks on the reformulation in Section 7.5 as applied on the inexact Uzawa algorithms in Section 7.6 and on the symmetric preconditioners in Section 7.7.
First the scalar products (7.14), (7.15), (7.22), (7.23) and (7.24), contain the preconditioners $\hat{A}$ and $\hat{C}$. If the preconditioners change during the iterative process it is almost impossible to say anything about the convergence. For changing preconditioners the scalar product should be independent of $\hat{A}$ and $\hat{C}$. Zulehner [Zul02] shows that it is possible to construct such D and its corresponding scalar product. A matrix E can be constructed such that it fulfills (7.11) with the prescribed matrix D. Here it is enough to know, that we prefer the preconditioners that are fixed during the iteration process, if we use these methods.

Secondly we remark the benefits of the reformulation. In the special cases, the original non-symmetric system is transformed into a symmetric system, with real eigenvalues. Because the eigenvalues have a positive real component, the system is positive definite too. For the symmetric positive definite system that we obtain by the reformulation, a conjugate gradient method can be used, that converges much faster than iterative methods for non-symmetric or indefinite systems.

The last remark is on the introduction of a new scalar product. This product is used in the convergence criterion which may have strange side effects. We illustrate our concern in the following example.
Suppose we use in a symmetric preconditioner for K, the following preconditioners for A and C

$$\hat{A} = \alpha A \quad \text{and} \quad \hat{C} = \beta C,$$

where we choose $\beta > 1$ and $\alpha = \frac{2}{1+\beta}$. This preconditioners are used in a symmetric preconditioner. Now the following inequality holds,

$$1 < \alpha^{-1} < \beta.$$

With this choice of the constants the conditions for a symmetric preconditioning algorithm as formulated in theorem 7.4 are satisfied:

$$\hat{A} = \alpha A \prec A$$

and

$$\hat{C} = \beta C \succ \alpha^{-1} C = \alpha^{-1} B A^{-1} B^T = B \hat{A}^{-1} B^T.$$

Let λ be an eigenvalue of A and let u_λ be the corresponding eigenvalue. Further let μ be an eigenvalue of C and p_μ the corresponding eigenvalue. Now we can examine the $\|.\|_D$ norm of (u_λ, p_μ),

$$\|(u_\lambda, p_\mu)\|_D^2 = \left\langle \begin{pmatrix} u_\lambda \\ p_\mu \end{pmatrix}, \begin{pmatrix} u_\lambda \\ p_\mu \end{pmatrix} \right\rangle_D = \langle (A - \hat{A}) u_\lambda, u_\lambda \rangle + \langle (\hat{C} - B \hat{A}^{-1} B^T) p_\mu, p_\mu \rangle.$$

If we look at the two parts separately, we see

$$\langle (A - \hat{A}) u_\lambda, u_\lambda \rangle = \langle (1 - \alpha) A u_\lambda, u_\lambda \rangle = (1 - \alpha) \lambda \|u_\lambda\|^2$$

and

$$\langle (\hat{C} - B \hat{A}^{-1} B^T) p_\mu, p_\mu \rangle = \langle (\beta C - \alpha^{-1} C) p_\mu, p_\mu \rangle = (\beta - \alpha^{-1}) \mu \|p_\mu\|^2.$$

The introduction of a new norm influences the convergence criterion. We get better preconditioners by letting $\beta \downarrow 1$, that implies $\alpha \uparrow 1$. A direct consequence is that the norm tends to zero. Therefore it seems possible that convergence in the new norm doesn't mean the same convergence in the old norm. Especially if some eigenvalues and eigenvectors of A and $\hat{A}$ or C and $\hat{C}$ are close to each other, the new norm can give a bad estimate for the real error in the direction of those eigenvectors. However more research is needed to determine how serious these troubles are. Maybe the problems can be solved easily by the formulation of preconditioner independent norms as in the first remark.

7.9 Grad-Div Stabilization

In [OR01] Olshanskii and Reusken propose to take the gradient of equation (6.2) and add it to equation (6.1). That means that the left-hand side of the first equation gets an extra term $\xi\nabla(\nabla\cdot u)$ with ξ positive and the right-hand side a term $\xi\nabla g$. This modification is called grad-div stabilization and it does not affect the solution.
The method can be seen as an *augmented Lagrangian* method. That means we try to minimize the residual of the variational formulation of equation (6.1) under the constraint of equation (6.2).

The modification does affect the discretized problem which now becomes

$$\tilde{K}\begin{pmatrix} u \\ p \end{pmatrix} \equiv \begin{pmatrix} \tilde{A} & B^T \\ B & 0 \end{pmatrix}\begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} \tilde{f} \\ g \end{pmatrix} \tag{7.27}$$

where

$$\tilde{A} = A + \xi B^T B \quad \text{and} \quad \tilde{f} = f + \xi B^T g$$

The new Schur-complement becomes

$$\tilde{C} = B\tilde{A}^{-1}B^T$$

In [OR01] the condition number of the Schur-complement is estimated

$$\kappa(\tilde{C}) \leq \hat{\beta}^{-2}\frac{\nu + \alpha c_F^2 + \xi}{\nu + \alpha c_I^{-2}h^2 + \xi}$$

with ν and α as in equation (6.4) and $\hat{\beta}$, c_F and c_I positive constants independent of h. For $\nu \to 0$ and $\xi = 0$ the bound of the condition number behaves like $O(h^{-2})$. The stabilizing effect of $\xi > 0$ is evident.

Because of the equality

$$\nabla(\nabla\cdot u) = \Delta u + \nabla \times \text{curl } u$$

grad-div stabilization can be explained by the addition of a diffusion term to equation (6.1). Troubles with the solution of this equation occur if ν tends to zero and the diffusion term is not dominant anymore. Grad-div stabilization adds a diffusion term (and a term $\nabla \times$ curl u) in order to benefit from its stabilizing properties.

It can be questioned how important grad-div stabilization really is. If ν get small, the diffusion term can be dropped and the momentum equation becomes $\alpha u + \nabla p = f$. We can simply substitute u in the continuity equation and solve $\Delta p = \nabla\cdot f - g$. This is an easy solvable Poisson equation for the pressure.

Olshanskii and Reusken use grad-div stabilization in combination with an inexact Uzawa algorithm. They choose for $\tilde{A}$ a standard multigrid V-cycle as preconditioner and for $\tilde{C}$ the simple preconditioner $\hat{C} = \gamma I$. Then $\kappa(\hat{C}^{-1}\tilde{C}) = \kappa(\tilde{C})$.

It is important that the preconditioner of $\tilde{A}$ includes the term $\xi B^T B$. It is not sufficient to use grad-div stabilization in residual calculations only.
For small values of ν grad-div stabilization improves convergence. The Stokes problem with $\nu \ll 1$ not often occurs in practice. Nevertheless this approach is important because this method has similar effects for (linearized) Navier-Stokes equations with high Reynolds numbers. In that case grad-div stabilization may have computational advantages, although according to [JMM+00] the memory requirements could raise considerably.

Chapter 8

Preconditioning

In the previous chapter we introduced some basic approaches for the saddle point problem. The inexact Uzawa algorithms and the symmetric preconditioners use preconditioners for the matrices A and C. In this chapter we introduce a number of methods to construct such a preconditioner.

Assume we want to solve the equation $Ax = b$. In general for large sparse systems direct computation of the solution $x = A^{-1}b$ is expensive. Therefore we choose a smart matrix $\hat{A}$ and transform the linear system into

$$\hat{A}^{-1}Ax = \hat{A}^{-1}b. \tag{8.1}$$

The matrix $\hat{A}$ is called the preconditioner. To make this preconditioning useful $\hat{A}$ has to fulfill the following conditions:

- $\hat{A}$ is easy to construct.
- $\hat{A}$ is such that the system $\hat{A}x = b$ is easy to solve.
- The eigenvalues of $\hat{A}^{-1}A$ are close to 1.

If $\hat{A}$ has this properties we can solve equation (8.1) in some iterative method like (F)GMRES or, if $\hat{A}^{-1}A$ is symmetric and positive definite, PCG and BiCGSTAB. For descriptions of those methods see Meurant [Meu99].

There are many ways to construct a preconditioner. In [Meu99] an overview of the basic methods is presented. We can use a repeated Red-Black factorization or for symmetric problems an incomplete Cholesky factorization. There exist polynomial preconditioners and multilevel ILU preconditioners. Many variants are derived from these preconditioners.
In this section just a few preconditioners are discussed. We will examine those because they were used in some solver for the saddle point problem or may be used in future.

8.1 Preconditioner for Matrices with Dominant Skew Symmetric Component

In [GV98] Golub and Vanderstraeten introduce a preconditioner for problems with a dominant skew symmetric component. The preconditioner is tested on the steady state Navier-Stokes equations and on an Oseen equation with $w \equiv 1$ and without the pressure term. We shortly introduce the preconditioner.

Let A be a non singular matrix. Consider the following splitting

$$A = H + S,$$

with $H = (A + A^T)/2$ hermitian and $S = (A - A^T)/2$ skew-symmetric.
A standard preconditioner based on this splitting is

$$\hat{A}^{-1} = (I - H^{-1}S)H^{-1}.$$

If the skew symmetric part is dominant this preconditioner is bad. The error is $\|A^{-1} - \hat{A}^{-1}\|_2 = O(\|H^{-1}S\|_2^2)$, that becomes large with dominant S. Therefore we do it the other way around and interchange H and S. Now the inverse of the dominant matrix, S^{-1} is used instead of H^{-1}. However to make sure S is invertible, the matrix needs a shift, that makes it positive definite. Therefore we introduce

$$\tilde{S}(\alpha) = S + \alpha I,$$
$$\tilde{H}(\alpha) = H - \alpha I,$$

with α a positive real number. The sum of both equals A.
Now consider the following preconditioner

$$\hat{A}^{-1}(\alpha) = (I - \tilde{S}^{-1}\tilde{H})\tilde{S}^{-1}. \tag{8.2}$$

Convergence is sure if

$$\rho((\tilde{S}^{-1}\tilde{H})^2) = |\mu_{max}|^2 < 1,$$

with μ_{max} the eigenvalue of $\tilde{S}^{-1}\tilde{H}$ with the largest modulus. The fastest convergence is obtained if μ_{max} is minimal. A good approximation of the minimum is found for

$$\alpha = \frac{\lambda_1 + \lambda_n}{2},$$

with λ_1 and λ_n respectively the smallest and largest eigenvalue of H.

This preconditioner requires the evaluation of $\tilde{S}^{-1}$ or equivalent we have to solve systems $\tilde{S}p = (S + \alpha I)p = q$. Solving the normal equations ($\tilde{S}^T\tilde{S}p = \tilde{S}^T q$) appears to be more effective than other methods despite the squaring of the condition number. Explicit computation of the normal matrix can be avoided. If we have an incomplete QR-factorization of $\tilde{S}$, the normal matrix can be approximated by $R^T R$. This results in a nested iteration procedure. The inner iterations solve the normal equations with use of the QR factorization. The outer iterations solve the main equation $Ax = b$ with use of preconditioner (8.2).

For some problems with dominant skew symmetric part this approach is succesful. In the modified Oseen equation and the steady-state Navier-Stokes equation the preconditioner is useful for low viscosity. A preconditioner can be constructed for some problems were for example ILUT can not. The choice of α is delicate. For small values of α few outer iterations are needed, but for large values the incomplete QR-factorization will be better, which results in less inner iterations.

8.2 Incomplete LU factorizations

A special class of preconditioners is based on the construction of an incomplete LU factorization. Even with an optimal ordering like Nested Dissection a standard LU factorization will generate a lot of fill in both L and U. Much fill costs memory and time, because the computation of $(LU)^{-1}$ will be expensive. To reduce the fill some elements are dropped during the construction of L and U. This results in an incomplete factorization like

$$\hat{A} = LU = A - E,$$

where E is the matrix that contains the dropped elements.

Suppose we have a subset $x_1 \subset x$ and further $x_2 = x \setminus x_1$. The subset must contain at least one point. If we want to eliminate x_1, we can reorder our system

$$\begin{pmatrix} A_{11} & A_{22} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

In all parts of A elements can be dropped such that we get for each A_{ij} an approximation $\hat{A}_{ij}$. With these approximations we can construct the Schur-complement in order to get the following incomplete LU factorization.

$$\hat{A} = \begin{pmatrix} \hat{A}_{11} & \hat{A}_{22} \\ \hat{A}_{21} & \hat{A}_{22} \end{pmatrix} = \begin{pmatrix} \hat{A}_{11} & 0 \\ \hat{A}_{21} & A^{(2)} \end{pmatrix} \begin{pmatrix} I & \hat{A}_{11}^{-1}\hat{A}_{12} \\ 0 & I \end{pmatrix}$$

with the Schur complement

$$A^{(2)} = \hat{A}_{22} - \hat{A}_{21}\hat{A}_{11}^{-1}\hat{A}_{12}$$

This process can be repeated with $A^{(2)}$ to get $A^{(3)}$ and so on.

Important choices for ILU methods are the dropping criteria (*drop-by-level* or *drop-by-size*) and the numbering of unknowns. The numbering determines the order of variables to be eliminated and has a large impact on the fill in the final LU factorization. Lumping of dropped elements to the diagonal, the *Gustafsson-modification*, usually results in a better factorization.

8.2.1 ILU(l)

This important ILU method is based on *drop-by-position*. The constant l denotes the level of fill in L and U with respect to the fill in A. ILU(0) allows fill just where A has fill. ILU(l) allows a higher level of fill.
There are two problems with this method. First it is difficult to determine for several problems and matrices where the extra fill is allowed. Second the ordering of unknowns has a large effect on the quality of the preconditioner. Which ordering is best depends on the problem and on the allowed level of fill.

8.2.2 MUM-ILU(l)

A minimum discarded fill (MDF) ordering for an ILU(l) factorization chooses the next pivot node such that the size of the discarded fill is minimized. The MDF ordering is effective for problems with a small molecule. However for sparse matrices with a large molecule the construction of this ordering is very expensive. MDF requires a search trough the nested linked list of A, which is a costly kind of searching. In order to avoid this search it is possible to use a cheaper ordering, the minimum update matrix (MUM) ordering. The idea of this ordering is to choose the next node such that the Frobenius norm of $\tilde{A}^{(2)}$ is minimized, where $\tilde{A}^{(2)}$ equals $A^{(2)}$ without the diagonal entries. If $\|\tilde{A}^{(2)}\|_F$ is small, the discarded fill entries must be small too, because they are a subset of the entries in $\tilde{A}^{(2)}$. For a detailed description of the algorithm see [CDFT92].

8.2.3 Matrix Renumbering ILU

Matrix renumbering ILU (MRILU) has *drop-by-size* and renumbering as basic principles. An extended discussion and some results of this method can be found in [BW99].
The idea is to search a nearly independent set of variables and eliminate them with dropping of all internal connections. This search is based on matrix entries only, so MRILU can be considered as black-box solver. A variable is added to x_1, the set that will be eliminated, if the connections to the existing set x_1 are small. The set it constructed such that the coefficients of A_{11} satisfy

$$\sum_{i \neq k} |a_{ik}| \leq \varepsilon |a_{ii}| \text{ with } \varepsilon < 1, \tag{8.3}$$

where ε denotes the drop tolerance.
From the equation it follows immediately that A_{11} is strongly diagonal dominant, so it can be replaced by its diagonal. Dropping outside A_{11} is limited to A_{12} and A_{21} and has to be done very carefully. The Schur-complement can be computed and the search for a new nearly independent set can be started.
MRILU can be easily generalized for the block approach of Section 7.1. Lumping becomes block lumping and the diagonal becomes a block diagonal.
This method is a multi-level method and can be compared with an algebraic Multigrid method with V-cycle and without smoothing.

8.3 Multigrid Methods

We will explain the basic principles of multigrid methods with an example of a two-grid method. Assume we have a fine grid Ω_h with mesh size h and a coarse grid Ω_H with mesh size H. The aim is to solve the following linear equation on Ω_h

$$A_h u_h = b_h.$$

Let u be the exact solution on the grid and let $e^k = u - u^k$ be the error. Then the equation can be rewritten

$$A_h e^k = A_h u - A_h u^k = b - A_h u^k = r^k.$$

Of course this equation is not easier to solve than the original. But if we know an approximation v^k of e^k then we have with $v^k + u^k$ probably a better guess for u. With multigrid such an approximation can be computed on the coarser grid Ω_H.

To transform solutions from Ω_h and Ω_H and back we need two operators.
A restriction operator

$$R : \Omega_h \to \Omega_H,$$

and a prolongation or interpolation operator

$$P : \Omega_H \to \Omega_h.$$

Classical methods as relaxed Jacobi or Gauss-Seidel are known to be good smoothers. A few iterations of those methods can be used to smooth the residuals. If the residuals are smooth enough it could be useful to switch to a coarser grid in order to smooth the residual on a larger scale.

All multigrid methods are based on the following algorithm taken from [Meu99].

Algorithm 8.1. *The two-grid method contains the following steps:*

1. *Pre-smoothing, do ν_1 iterations of the iterative method whose iteration matrix is S let $\bar{u}^k$ be the resulting vector.*
2. *Calculate residual, $\bar{r}^k = b - A_h \bar{u}^k$.*
3. *Restriction, $\bar{r}_H^k = R\bar{r}^k$.*
4. *Solve system on coarse grid, $A_H e_H^k = \bar{r}_H^k$.*
5. *Prolongation (or interpolation) $v^k = Pe_H^k$.*
6. *Post-smoothing, starting from $\bar{u}^k + v^k$ do ν_2 iterations of the method whose matrix is S let u^{k+1} be the result.*
7. *If no convergence go to step 1.*

In this algorithm there are many choices still to be made. We have to find a smoother or relaxation method S, choose integers ν_1 and ν_2, find a way to construct the coarse grid, the restriction operator, the prolongation operator and we have to define A_H. All aspects should reduce the spectral radius of the iteration matrix

$$M = S^{\nu_2}(I - PA_H^{-1}RA_h)S^{\nu_1}.$$

In many cases the prolongation operator is chosen the transpose of the restriction operator, so

$$P = R^T.$$

It is easy to expand this method to more than two levels. In that case we have to choose how we walk through the levels. If we go from the finest to the coarsest and back we get a V-cycle. Another popular choice is the W-cycle, that allows some extra V-cycles on the coarser levels before one returns to the finest level.
In the multigrid method one starts with an initial solution on the finest grid. A variant of multigrid called *full multigrid* (FMG) starts on the coarsest grid with an exact solution. This initial solution is prolongated to the finer grids.

The multigrid methods can be divided in two groups. The geometric multigrid solvers choose easy grids and search for smart smoothers. The algebraic multigrid solvers use easy smoothers and search for smart grids.

8.4 Geometric Multigrid

The original multigrid methods use information about the geometry of the problem to construct a sequence of grids in advance. Therefore they are called geometric multigrid methods. In many cases the sequence of grids is simply constructed by halving the mesh size in an initial coarse grid. Because the grids are given, the only possibility to improve the method is to choose smarter smoothers. Simple smoothers like Gauss-Seidel or relaxed Jacobi have shortcomings at complicated problems. There are several alternatives like symmetric coupled Gauss-Seidel (SCGS) [Van86], Vanka-type smoothers [JT00] which can be considered as block Gauss-Seidel methods, or the transforming smoothers of Wittum [Wit90]. Since the last smoothers were designed for (Navier)-Stokes, we pay attention to them.

8.4.1 Transforming Smoothers

A splitting $A = G - H$ of A is called *regular* if G is monotone and $H \geq 0$ or *weak regular* if G is monotone and $G^{-1}H \geq 0$. G being monotone is equivalent with G non-singular and $G^{-1} \geq 0$. Here $G^{-1} \geq 0$ means that all matrix entries of G^{-1} are positive.
The saddle point problem is indefinite so a (weak) regular splitting is not possible. A transformation could change this property. Let T_L and T_R be nonsingular transformation matrices such that we are able to produce the following weak regular splitting

$$T_L A T_R = G - H. \tag{8.4}$$

Now we have also a splitting for A

$$A = T_L^{-1} G T_R^{-1} - T_L^{-1} H T_R^{-1}.$$

The smoother/iteration matrix that is related to the transformed iteration is given by

$$S = T_R(G^{-1}H)T_R^{-1}. \tag{8.5}$$

For convergence $\rho(S) \leq 1$ is sufficient. If $T_L = I$ we have an R-transforming smoother.

For practical reasons the smoother in equation (8.5) is often applied as perturbed transforming smoother. That means some inconvenient terms in the product system are left out the transformation.

8.5 Algebraic Multigrid

Extended information about algebraic multigrid (AMG) can be found in [Stü99]. The advantage of this method is, that it can be used as a black-box solver for partial differential equations. This in contrast with geometric multigrid where a lot of information about the problem is required in order to choose the right coarser grids.
The main idea of algebraic multigrid is that the construction of a coarser grid is related to the smoothness of the solution after a few steps with the smoother. In areas where the error is very smooth we can use a local coarser grid than in areas where the error is not smooth at all. AMG determines itself what the next grid will be. So we can use for difficult areas in the flow finer grids than in other regions.

Application of AMG to a problem is a two part process. The first part is the *setup phase*, that consists of choosing the coarse levels and defining the transfer and coarse-grid operators. The second part, the *solution phase* uses those grids and operators in normal multigrid cycling (described for two-grid in algorithm 8.1) until a desired level of accuracy is reached.

An error is called *algebraically smooth* if the convergence with respect to the smoother S is slow, or equivalent $Se \approx e$. In the direction of algebraically smooth error it is possible to coarsen the grid.

Let Ω denote the set of points of the grid. We want to split Ω in two disjunct subsets F (the fine grid points) and C (the coarse grid points). The coarse grid should be small to reduce cost of computations on the coarse level. On the other hand the F-to-C connectivity should be strong in order to get fast convergence. There are several ways to do this coarsening. We will pay attention to the standard coarsening as proposed in [Stü99].

Define a variable i to be strongly negatively coupled (n-coupled) to variable j, if

$$-a_{ij} \geq \varepsilon_{str} \max_{a_{ik}<0} |a_{ik}| \quad \text{with fixed} \quad 0 < \varepsilon_{str} < 1.$$

Denote the set of all strong n-couplings of variable i by S_i with

$$S_i = \{j \in N_i : i \text{ strong n-coupled to } j\},$$

and the strong transpose n-couplings by S_i^T with

$$S_i^T = \{j \in \Omega : i \in S_j\}.$$

We use these notations in the following algorithm obtained from [Stü99].

Algorithm 8.2. *Algorithm for standard coarsening in Algebraic Multigrid. Here U denotes the set of undecided points, F the set of fine grid points and C the set of coarse grid points. For any set P, $|P|$ is the number of elements in the set.*

1. *Start with* $F := \emptyset, C := \emptyset, U := \Omega$.
2. *Compute* $\lambda_i = |S_i^T \cap U| + 2|S_i^T \cap F| \ \ (i \in U)$.
3. *If* $\lambda_i = 0$, *finished.*
4. *Else, pick* $i \in U$ *with max.* λ_i: $C := C \cup \{i\}, U := U \backslash \{i\}$.
5. *For all* $j \in S_i^T \cap U$: $F := F \cup \{j\}, U := U \backslash \{j\}$.
6. *Update* λ_i *and go back to step 3.*

This algorithm just uses direct couplings for coarsening. All variables in F have at least one direct coupling to C. For problems with small molecules this leads to relative large coarse grids. In *aggressive coarsening* the definition of strong connectivity is extended to variables that are not directly coupled. A variable i is said to be strongly n-connected to a variable j *along a path of length* l if there exist a sequence $i_0, i_1, \ldots, i_l$ with $i = i_0$ and $j = i_l$ such that $i_{k+1} \in S_{i_k}$ for $k = 0, 1, 2, \ldots, l-1$. With given values of $p \geq 1$ and $l \geq 1$, we then define a variable i *to be strongly n-connected to a variable* j *w.r.t* (p, l) if at least p paths of length l exist such that i is strongly connected to a variable j along each of these paths. Now S_i in algorithm 8.2 can be replaced by

$$S_i^{p,l} = \{j \in \Omega : i \text{ strongly n-connected to } j \text{ w.r.t. } (p, l)\}.$$

For small molecules aggressive coarsening can be used in combination with standard coarsening. Then aggressive coarsening is only used to construct the second level. On that level the molecules will be larger and we can use standard coarsening to construct higher levels.

For the algorithm we assumed the existence of dominant negative couplings. In case of strong positive connections some subtile changes are needed in the algorithm. For details see [Stü99].

8.6 Discussion

In this chapter and the previous one we discussed several reformulations and solving techniques for the saddle point problem. Now we would like to answer the important question: what method provides the best solver? Unfortunately the answer is not as simple as the question.

For the comparison of different methods we take in account three aspects

- memory requirements,
- convergence behavior,
- required floating point operations (flops).

Important is the convergence behavior. The best we may expect is grid independent convergence. That means, if n is the number of unknowns, then the total amount of work is $O(n)$. Note that grid independent convergence is very important for large n. The best direct methods for 2D-regular grids are $O(n\sqrt{n})$ and for 3D grids $O(n^2)$ which causes a considerably growth in computation costs if n is doubled a few times.

A fair comparison of methods is difficult, because of lack of a direct comparison of all solvers. In most papers the author proposes a method and shows some results of that method for a specific problem. Tested problems of saddle point type are for example the Oseen equation [Ols99] or Stokes with variable coefficients and mixed boundary conditions [BP88], linearized Navier-Stokes with polynomial solution [JMM+00], the steady state driven cavity problem [BW99] [CDFT92] [Van86] [Wil96] [Zul02] or the flow around a wing [Mav95] [PK94].
Even if two methods handle the same problem a good comparison based on papers is almost impossible. The performance of a solver is expressed in total time with or without the construction of the preconditioner, the total number of flops or the number of iterations in a CG or Newton method. Each author uses his own standard. However we will try to evaluate the quality of the solvers and draw some conclusions.

Which solver is best for the saddle point problem depends on the choice of the parameters α and ν and the vector w in equation (6.4). These parameters influence directly the properties, for instance symmetry and definiteness, of A and B and therefore also of C and K.
Based on the results of the solvers in previous sections as presented in the referred papers we draw the following conclusions.

1. In case of a *symmetric* and *positive definite* matrix A, the inexact Uzawa algorithms seem the best basic iterative methods. With well chosen preconditioners for A and C, a symmetric positive definite reformulation is possible, so a conjugate gradient method can be used. Even with a simple preconditioner for C fast convergence is obtained.

   The same holds for the class of symmetric preconditioners. However the extra correction step for the velocity doubles the computational costs per iteration, while the convergence rate is almost equal to that of the inexact Uzawa algorithms.

2. The case that A is *nonsymmetric* or *indefinite* is more complicated. The convergence analysis on the reformulation of the inexact Uzawa algorithms and the class of symmetric preconditioners, that is heavily based on the symmetry and positive definiteness of A and C, is not valid anymore. The block approach seems a serious candidate in this case. More research and convergence analysis is required for this case.

3. In all cases a completely separated approach like in the Schur-complement method (see Section 7.2) seems not very useful. For several problems equation (7.1) has to be solved very accurate. In [JMM+00] this is not very important, but still the Schur-complement method is beaten by a geometric multigrid method for the block approach (see Section 7.1) of the saddle point equation.

4. Grad-div stabilization is a potential improvement if the viscosity ν gets small with respect to α. In [JMM+00] Grad-div stabilization is used in combination with a block-diagonal preconditioner for the linearized Navier-Stokes equation, which causes an increase in memory usage and bad convergence. On the contrary in [OR01] Grad-div stabilization with inexact Uzawa and a multigrid method leads to grid independent convergence for the generalized Stokes equations. So grad-div stabilization can be useful if it is used in combination with appropriate preconditioning.

5. We reject the preconditioner for matrices with a dominant skew-symmetric component (see Section 8.1) as a serious candidate for the saddle point problem. A large convective term in equation (6.4) causes a serious skew-symmetric component in A, but according to the results in [GV98] the skew-symmetric part has to be very dominant in order to make the method perform better than commonly used other methods.

6. Among the preconditioners in this chapter the multigrid methods show the best performance. They can be used for the block approach [JMM+00], but also in combination with an inexact Uzawa algorithm. Occasionaly grid independent convergence is obtained. In general algebraic multigrid is better than geometric multigrid. The only multilevel method that seems to be able to compete with algebraic multigrid is matrix renumbering ILU, which is also a black box solver.

We have drawn several conclusions, but as usually with every answer many new questions arise. Therefore at the end of this literature study we point out directions for further research at problems of saddle point type.

1. First of all saddle point problems with nonsymmetric or indefinite A deserve examination. The exposition in Chapter 7 is mainly restricted to symmetric and positive definite case. Are inexact Uzawa algorithms and the symmetric preconditioners useless in this case? Especially the effect of the coriolis terms is of interest.

2. Another research subject is the effect of the reformulation and the introduction of a new scalar product. How serious is the criticism mentioned in the final remark in Section 7.8?

3. Finally the methods presented in this chapter and the previous one are meant to be used in practical numerical problems. So it is interesting to do some numerical experiments on some standard problems and compare the results of several methods, especially because there is lack of a good comparison.

Chapter 9

Fill Reduction in Multilevel ILU Methods

In the previous chapter we discussed a number of preconditioners found in the literature. In this chapter a preconditioner is added to the list. We introduce the solver and show some test results, but first we motivate the ideas behind the preconditioner.

9.1 Motivation

We develop a method along the lines of the block approach as described in section 7.1. In fact we want to improve MRILU (see section 8.2.3) for fluid flow problems, such as the saddle point problems in section 6.1. Although MRILU can solve fluid flow problems, we experience a too high fill on the coarser levels which slows down the factorization process.
For exact LU factorizations the ordering of unknowns has a large impact on the fill. The fill has a direct influence on the total amount of work (number of flops) in the iterative method. The effect of the ordering in direct methods is shown in Table 9.1, that is copied form [Bot98]. We see that the fill (nnz(L)) increases with the number of variables, but in case of a random ordering it does so much faster than with a smart ordering like nested dissection. The total work (flops) needed to construct the LL^T factorization reacts even more intens on an increase of variables.

We suppose that the fill in multilevel ILU methods can be reduced by the incorporation of fill-reducing ordering ideas of direct methods. This has the additional advantage that we end up with a direct method if the drop tolerance tends to zero.
The aim of this chapter is the examination of the supposition. To study the fill behavior we constructed in MATLAB a multilevel ILU factorization based on a nested-dissection ordering. In the next sections nested dissection is introduced, the dropping strategy is explained and finally the method is tested for two problems, the Laplace equation and the Stokes equation.

Ordering	flops/1000				nnz(L)/1000			
$N =$	100	400	1600	6400	100	400	1600	6400
Random	35	968	78944	4477865	1.5	14	216	3110
Rev. Cuthill-McKee	7	96	1410	21510	0.8	6	45	351
Nested-dissection	7	78	804	7636	0.8	5	28	153
Minimal Degree	5	53	590	7337	0.7	4	22	126

Table 9.1: Effect of ordering on the amount of work and memory for LL^T factorization of Laplace problem on a regular square grid.

9.2 Nested Dissection

In this section we show how a nested-dissection ordering can be used to construct an exact LU factorization.
Assume we have a grid that can be divided in a number of cells. A *cell* is a group of points that are all connected to each other. A point of the grid can be part of one or more cells. By definition all connections in a grid are internal connections of some cell. Two cells can have a set of common points, which is called a *separator*. If a separator is eliminated all points in the two cells will become connected and they merge to one new cell.
If a nested-dissection ordering is given, then an LU factorization can be constructed by elimination of separators. In Figure 9.1 the process of elimination is depicted. The upper left figure shows a part of the grid in starting position. This grid can be obtained from a regular grid with a nine-points molecule if we eliminate a quarter of the points in the center of the cells. The first grid contains square cells with one point on each separator. The white points in the next plot shows the separators to be eliminated. The lower left plot shows the separators to be eliminated thereafter, which brings us in the final plot to the starting position on a coarser level.

This process of eliminating separators can be repeated until the last cross is eliminated. By each elimination the number of cells is halved, but meanwhile the number of unknowns on the separators is doubled. For the elimination of separators the inverse of a block-diagonal matrix has to be computed. The blocks are of the same size as the separators. So the elimination process gets more expensive on a higher level. The elimination of the last largest separator will be by far the most expensive.
On a regular grid the construction of an LU factorization with a nested-dissection ordering is known to be of optimal order. For a more extended introduction into nested dissection and other orderings see [Meu99]

9.3 ILU Based on Nested Dissection

The construction of the exact LU factorization as described in the previous section, is expensive in time. This is mainly caused by the increasing size of the separators. Therefore we would like to reduce somehow the number of variables on the separators during the construction of the LU factorization.
This can be done effectively in the following way. Between two levels of exact elimination, we search for a set of points on the separators that can be eliminated approximately. Important condition is the maintenance of the cell structure of the grid. Hence new connections between points in different cells are not allowed and if such connections appear, then they must be thrown away immediately. In the next section we pay attention to the dropping strategy.
After the reduction of unknowns in the separators we can resume the construction of the LU factorization as sketched in the previous section. So we get a method for the construction of an incomplete LU factorization that mixes exact and approximate elimination. We will refer to this method as *Nested-Dissection ILU*, shortly NDILU.

9.4 Dropping Strategy

In this section we focus on the dropping strategy used in incomplete nested-dissection. We follow basically the same procedure as in section 8.2.
Suppose we have a set x_1 that is a subset of x. x_1 is candidate to be eliminated with dropping. Define $x_2 = x \setminus x_1$ and reorder the system

$$\begin{pmatrix} A_{11} & A_{22} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}. \tag{9.1}$$

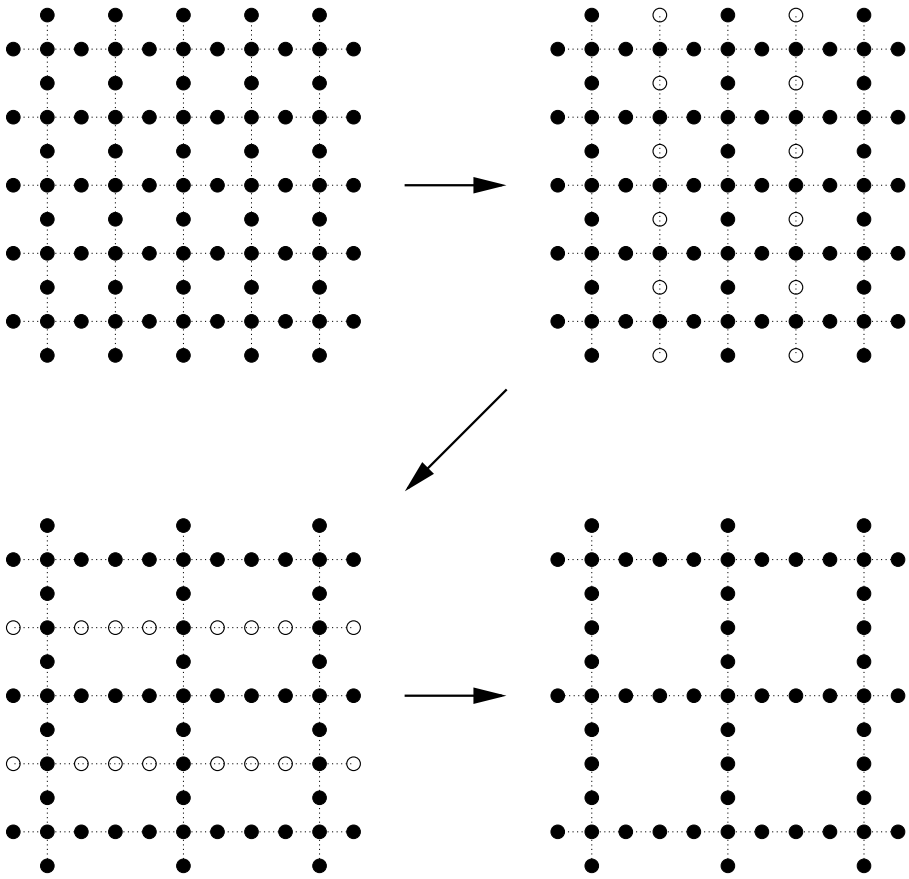

Figure 9.1: Elimination of separators in nested-dissection ordering on regular grid.

First we drop all entries outside the diagonal in A_{11}. So each point in x_1 is just connected to itself and points in $x_2 = x \setminus x_1$. We can write $A_{11} = \hat{A}_{11} + E_{11}$ with $\hat{A}_{11}$ a (block) diagonal matrix and E_{11} the error matrix. This approximation is used in the Schur-complement,

$$S = A_{22} - A_{21}\hat{A}_{11}^{-1}A_{12}.$$

If a point of x_1 is part of two different cells, the elimination of this point will connect in general all points of these two cells. We wanted to maintain the cell structure of the grid, so we have to drop these new connections immediately. So we get

$$S = \hat{S} + E_{22},$$

where E_{22} contains the undesired new connections.

Figure 9.2 illustrates the process of elimination and dropping for a fragment of the grid. We want to eliminate the white points, but meanwhile maintain the cell structure of the grid. So the points of the left cell are not allowed to be connected to the points in the right cell, except the separator of course.

First the internal connections, depicted by the dotted lines (1), are dropped. This prevents that all points of the cells in horizontal direction become connected if the set is eliminated. The breaking of these connections corresponds to the approximation of A_{11} with the diagonal matrix $\hat{A}_{11}$.

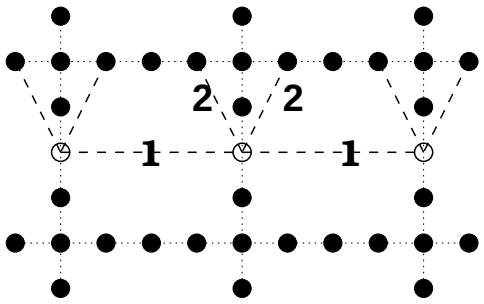

Figure 9.2: Dropping of connections.

The white points are still connected to points in two different cells, which is shown by the dotted lines (2). Elimination of the set will connect all points of the left cell to all points of the right one, which is not allowed, because the cell structure must be maintained. Therefore the newly created connections are dropped immediately, which corresponds to the dropping in the Schur-complement S.

Now we have an incomplete LU factorization for A:

$$A = LU + E = \begin{pmatrix} I & 0 \\ A_{21}\hat{A}_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} \hat{A}_{11} & A_{12} \\ 0 & \hat{S} \end{pmatrix} + \begin{pmatrix} E_{11} & 0 \\ 0 & E_{22} \end{pmatrix}.$$

We use lumping to the diagonal, so dropped elements are added to the diagonals of $\hat{A}_{11}$ and $\hat{S}$. By lumping the rowsum of E becomes zero and the method is exact for a constant solution. In case of more than one varable per grid point, this lumping is replaced by block-lumping.
The process of eliminating separators and dropping connections is repeated with $\hat{S}$. New dropping in this matrix can be added simply to E_{22}.

We want the set x_1 to be eliminated only if the dropping does not affect the condition number of A too much. Let D be the (block) diagonal of A. If the entries of E are small with respect to the entries of D, then in general the eigenvalues of LU and A will lie close to each other. Now inspired by equation (8.3) we formulate the following dropping criterion. The set x_1 is eliminated only if

$$\forall i : \sum_j |D^{-1}E|_{ij} < \varepsilon \text{ and } \sum_j |ED^{-1}|_{ij} < \varepsilon, \tag{9.2}$$

where ε denotes the drop tolerance. If we have more than one variable per grid point, D is a block-diagonal matrix, with block size equal to the number of variables. Since computations

with a diagonal matrix are much cheaper than with a block diagonal matrix, we compute an approximation for the left-inverse (maximum over column) and right-inverse (maximum over row),

$$D^c = \max_i |D^{-1}|_{ij} \text{ and } D^r = \max_j |D^{-1}|_{ij}.$$

Now define

$$S^c = D^c \, |A - D| \text{ and } S^r = |A - D| \, D^r. \tag{9.3}$$

Because we remove by the subtraction the block diagonal, S^c and S^r just contain the weighted off-diagonal elements of A. We can split S^c and S^r in four parts as we did with A in equation (9.1).

If we have

$$\forall i : \sum_j (S^c_{11})_{ij} < \varepsilon \text{ and } \sum_j (S^r_{11})_{ij} < \varepsilon, \tag{9.4}$$

then equation (9.2) holds for the part of E_{11}. If x_1 does not satisfy this condition the set can not be eliminated and we have to search for another set. However if x_1 satisfies the condition, we still have to check the effect of dropping in the Schur-complement. Therefore we compute estimates for $(S - A_{22})D_{22}^{-1}$ and $D_{22}^{-1}(S - A_{22})$, i.e. the part of the Schur-complement that originates from the elimination and that causes new fill in S where A_{22} has no fill. The entries are weighted with respect to the diagonal of A_{22}. We have the estimates

$$|SD_{22}^{-1}| = |A_{21}\hat{A}_{11}^{-1}A_{12}D_{22}^{-1}| = |A_{21}D_{11}^{-1}A_{12}D_{22}^{-1}| \leq |A_{21}D_{11}^{-1}||A_{12}D_{22}^{-1}| \leq S^r_{21}S^r_{12} \equiv K^r,$$

$$|D_{22}^{-1}S| = |D_{22}^{-1}A_{21}\hat{A}_{11}^{-1}A_{12}| = |D_{22}^{-1}A_{21}D_{11}^{-1}A_{12}| \leq |D_{22}^{-1}A_{21}||D_{11}^{-1}A_{12}| \leq S^c_{21}S^c_{12} \equiv K^c.$$

Note that S^r_{21} does not involve the zero diagonal in (9.3). Therefore we have

$$|A_{21}D_{11}^{-1}| \geq |A_{21}||D_{11}^{-1}| \geq |A_{21}|D^r_{11} = S^r_{21}.$$

A similar argument holds for S^r_{12}, S^c_{12} and S^c_{21}, so the above estimates are correct.
To get an estimate for the elements in E_{22}, we have to consider these entries in K^r and K^c corresponding with entries in the Schur-complement where no fill is allowed. Let F be the matrix that has ones where fill in the Schur-complement is allowed and zeros where no fill is allowed. If we have ('.*' means element wise multiplication)

$$\sum_j (K^c - F. * K^c)_{ij} < \varepsilon \text{ and } \sum_j (K^r - F. * K^r)_{ij} < \varepsilon, \tag{9.5}$$

then equation (9.2) is satisfied for the part of E_{22}. So if both equation (9.4) and equation (9.5) are satisfied the set x_1 can be eliminated.

9.5 Computational Aspects

We have written a program in MATLAB that constructs an incomplete LU factorization with use of the dropping strategy of the previous section. Between two levels of exact elimination we search for a nearly independent set a number of times.
The advantage of the formulations and estimates above, is that we have to compute S^c and S^r just once in the search for a set that can be eliminated. If we want to check another set we just have to make a new partition of the matrices. The first check in equation (9.4) can be done easily. The second check in equation (9.5) is more difficult, because K^c and K^r and the allowed-fill matrix F have to be computed. However we can choose the sets smartly. For example on a regular grid the middle points of the separators are most likely to generate the smallest entries in E_{22}. Therefore this set should be checked first.
In the next sections we test the algorithm for two problems. The resulting incomplete LU factorization will be used in a GMRES iteration process. The number of iterations needed to reach the desired accuracy is an indication of the condition number of the preconditioned matrix and the quality of the preconditioner.

	NDILU				MRILU		
ε	exact levels	approx levels	fill	iter	levels	fill	iter
0.6	0	14	1.6	27	-	-	-
0.3	5	14	2.0	17	8	1.3	47
0.1	11	16	2.9	13	17	2.4	11
0.03	13	46	4.3	9	31	3.9	7
0.01	13	56	6.1	6	44	5.1	5
0.003	13	58	7.6	4	61	6.9	4
0.001	13	47	9.3	3	80	8.0	3
0	13	0	11.4	1	-	-	-

Table 9.2: Results of NDILU and MRILU for the two-dimensional Poisson equation.

9.6 Results for the Poisson Equation

In this section we show some results of ILU based on nested-dissection for the two-dimensional Poisson problem. For this problem fast solvers are available. We do not really expect NDILU to beat these solvers, but we use the Poisson problem only as a first test problem. If NDILU is not able to produce a good preconditioner for the Laplace operator, it is unlikely that it will be able to do so on other problems.

The Poisson equation:

$$\Delta u = f \text{ in } \Omega \tag{9.6}$$

We take $\Omega \subset \mathbb{R}^2$ and use the standard central discretization of Δu. So we get a five-point molecule, that allows us to eliminate half the points immediately and we end up with a nine-point molecule. With this molecule we build a square grid of $(2^7 + 1) = 129$ points on each side. The total number of unknowns is 16641. Furthermore we use a Dirichlet boundary.

For several values of the drop tolerance ε, that is used in equations (9.4) and (9.5), a preconditioner is constructed with NDILU. The results are compared with the performance of MRILU (see Section 8.2.3) on the same original matrix. The comparison is presented in Table 9.2. In this table *fill* denotes the fill of the LU factorization relative to the fill of the original matrix, *iter* is the number of iterations in GMRES to reach an accuracy of 10^{-6} and *levels* denotes the number of sets that is eliminated by approximation. For the NDILU factorization there is a distinction between the levels of *exact* and *approximate* elimination.

From the table we can conclude that it is possible to construct a good preconditioner with ILU based on a nested-dissection ordering. NDILU reaches the results of MRILU. For the same level of fill almost the same number of iterations is needed. NDILU needs a little higher drop tolerance. An example of the output of the MATLAB-program for the construction of the preconditioner with NDILU for $\varepsilon = 0.1$ can be found in the appendix.

9.7 Results for the Stokes Equation

The second test involves the Stokes equation. It is a more difficult and more interesting problem than the previous one, because it is a fluid flow problem. The stokes equation is a saddle point problem as discussed in Chapter 6. For MRILU these type of problems are hard to handle.

	NDILU				MRILU		
ε	exact levels	approx levels	fill	iter	levels	fill	iter
0.9	9	28	3.0	286	-	-	-
0.8	10	32	3.5	146	-	-	-
0.7	10	29	3.5	119	-	-	-
0.6	10	38	3.8	53	18	2.2	x
0.5	10	28	3.2	73	15	3.1	90
0.4	11	36	4.5	29	17	3.4	29
0.3	11	41	5.5	23	15	6.4	14
0.1	11	30	9.8	8	24	8.9	8
0.03	11	21	10.4	3	29	12.9	5
0.01	11	3	10.5	2	44	15.0	4
0.001	11	0	10.5	1	49	19.8	3
0	11	0	10.5	1	-	-	-

Table 9.3: Results of NDILU and MRILU for the Stokes equation.

The Stokes equation

$$-\nu\Delta u + \nabla p = f \text{ in } \Omega, \tag{9.7}$$
$$\nabla \cdot u = 0 \text{ in } \Omega. \tag{9.8}$$

We use again $\Omega \subset \mathbb{R}^2$. The variables are the two-dimensional velocity vector u and the pressure p. To construct the grid and its corresponding matrix we use the same procedure as for the two-dimensional Poisson problem: (central) discretization, elimination of one half of the points to get a nine-points molecule, construction of a square grid of this molecule and Dirichlet boundaries. After discretization we scale the variables, such that they are all of the same order of magnitude. In each direction we have $2^6 + 1 = 65$ points with three variables per point. This brings the total number of unknowns at 12675.
Again we construct for different values of ε an incomplete factorization with NDILU. During the construction we use block lumping. That means that if blocks are dropped they will be lumped to the main block diagonal. We do the same for MRILU and test the quality of both decompositions with GMRES.
An example of the output of the MATLAB-program for the construction of the preconditioner with NDILU for $\varepsilon = 0.1$ in case of the Stokes equation can be found in the appendix.

The results are presented in Table 9.3. Dropping appears to be more critical for the Stokes problem. The drop tolerance has to be quite large before dropping has effect. For those large values of ε MRILU and the nested-dissection ILU compete with each other. But, more important, if ε gets small and even tends to zero, the fill generated by MRILU will rise fast and even get larger than the fill generated by an exact method. Meanwhile the fill of NDILU has a clear limit. If no set can be dropped NDILU will simply generate the exact decomposition. The fill will be always at most that of the exact LU decomposition based on a nested-dissection ordering.

9.8 Conclusions and Recommendations

In this chapter we introduced a new variant of the ILU factorization combining some properties of exact and incomplete factorizations. Based on the results in the previous two sections we draw the following conclusions.

1. The first and most important conclusion: it is possible to construct a useful preconditioner with an ILU factorization based on a nested-dissection ordering.

2. Unless the restrictions during the construction the quality of the preconditioner is comparable with that of MRILU for both the Poisson and Stokes equation.

3. Scaling of variables in the Stokes equation is important for NDILU as well as MRILU.

4. The desired fill reduction occurs only for low drop-tolerances. The ILU factorization based on nested dissection tends to the exact LU factorization with a maximum fill that equals the fill of the exact factorization. In this case the preconditioner of MRILU generates substantially more fill than the exact factorization.

The research in this chapter is a pilot project. We did some first explorations in the area of incomplete LU factorization based on fill-reducing orderings in direct methods. The results are promising, therefore we do some recommendations for further research.

1. First of all some research could be done on the release of the strict clustering of variables. The inexact Uzawa algorithms (see Section 7.3) use a separately treatment of the variables, which is successful for the Stokes problem. If the clustering of variables is released maybe the inexact Uzawa algorithms appears to be a special case of an incomplete LU factorization for the Stokes problem.

2. The second interesting research subject is the scaling. Appropriate scaling improves the performance of both NDILU and MRILU. Is there a relation between the clustering of variables and the importance of scaling?

3. Even if we stick at the clustering of variables many improvements are possible. The preconditioner was constructed under a large number of restrictions, that may be released. We mention some of the possible improvements.

   (a) In the present program, the search of an independent set is limited to the points on the separators and excludes the corner-points of the cells. Especially for the Poisson problem this limits the performance of the preconditioner. For $\varepsilon = 0.6$ in Table 9.2 the method reaches its maximum dropping, that means on each level the cells contain only corner-points. The internal points on a separator, that occur after merging of cells, are eliminated immediately. The fill will never get below 1.6, but MRILU shows that with less fill we still can produce a good preconditioner. Allowing corner-points to be eliminated too makes it possible to reduce more fill.

   (b) The second possible improvement involves the demands on the grid. In Section 9.4 we sticked strictly to the nested-dissection ordering of the grid. New connections were not allowed and dropped immediately. Maybe there is room to allow the existence of some connections without largely disturbing the properties of the nested-dissection ordering.

   (c) For the Stokes equation NDILU has a relative large number of levels. This is caused partially by the way the sets are selected. On each separator at most one point is part of the candidate set. If it is allowed to select more than one point in a time on each separator, the number of levels can be reduced.

   (d) Finally the boundary can be treated smarter. In the present code the points on the boundary are treated as if they were separators. If the last internal cross is eliminated we still have the points on the boundary, that are all connected to each other. However the points on the boundary, that are part of only one cell and not a separator at all, should be eliminated first. This is important especially if a few independent sets can be eliminated and we end up with many points on the boundary. So for low drop-tolerances the fill of NDILU in Tables 9.2 and 9.3 can be reduced.

4. Until now we neglected an important aspect in iterative methods: time. The preconditioner generated by an ILU factorization on a nested-dissection ordering may be very good, but if it takes a lot of time to construct this preconditioner, the method is useless. In time

MRILU beats NDILU by far, however this comparison between both methods is not fair. The NDILU program is in a premature state and written in MATLAB, whereas MRILU is sophistically programmed in the much faster language FORTRAN.
Especially the selection and rejection of candidate sets and the computation of the fill matrix are time-consuming elements in the algorithm. Fortunately the time spent on these elements can be reduced considerably with a few smart choices.

5. We examined the performance of the preconditioner on a regular square grid. A generalization to irregular grids is interesting. Maybe it is possible to transform the method into a black-box method. Based on the graph of a sparse matrix it is possible to construct a nested-dissection partition. Possibly the algorithm can be generalized to produce an incomplete LU factorization based on a generated nested-dissection partition. A few algorithms that produce such a partition are described in [Meu99].

Part III

Chapter 10

Conclusions

The last chapter of this thesis contains of course the inevitable conclusions. However we have already drawn conclusions at the end of both parts, so we confine ourselves to a short repetition. A more detailed exposition can be found in Section 5.5 for part I, in Section 8.6 for the literature study and in Section 9.8 for the fill-reduction in multilevel ILU methods.

In the first part of the thesis we examined two improvements of the numerical continuation code used by the IMAU: automatic step size control and the implementation of some variants of the Newton method. The adaptive Shamanskii method appears to be the best method among the alternatives for Newton. Step size control and the adaptive Shamanskii method are successfully implemented in the program code for numerical continuation of ocean circulation. The total speed up is about a factor 4.

In the second part we focused on the solver for the systems that occur in the ocean calculations of the IMAU. The relevant fluid flow problem can be viewed as a saddle point problem. In literature we found several approaches for these problems. Most important alternative for the approach of MRILU is the class of inexact Uzawa algorithms, that allow for a symmetric and positive definite reformulation in case of a symmetric problem. Unfortunately these algorithms are less useful in case of a non-symmetric or indefinite problems.
Finally we examined a new variant of the ILU preconditioners, that incorporates fill-reducing ordering ideas of direct methods in an incomplete LU factorization. Based on a nested-dissection ordering it is possible to construct a useful incomplete LU factorization that is able to compete with MRILU for Poisson and Stokes problems. Especially on the last subject, incorporation of fill-reducing ordering ideas in multilevel ILU methods, there is more research to be done.

Appendix A

Parameters in the Ocean Circulation Problem

For the ocean circulation problem we used the program for numerical continuation (THCM) that contains a number of parameters. We used the following configuration for the area, boundary conditions, grid and physical constants.

The area and the boundary conditions:

```
xmin  =  290. * pi/180.
xmax  =  350. * pi/180.
ymin  = -70.  * pi/180.
ymax  =  60.  * pi/180.
hdim  =  4000.
zmin  = -1.
zmax  =  0.

SLIP =  1
TRES =  1
SRES =  0
```

The grid:

```
n    = 12
m    = 28
l    =  6
nun  =  6
la   =  0
```

The physical constants:

```
omegadim = 7.292e-05
r0dim    = 6.37e+06
udim     = 0.1e+00
gdim     = 9.8e+00
rhodim   = 1.0e+03
t0       = 15.0
s0       = 35.0
cp0      = 4.2e+03
alphaT   = 1.0e-04
alphaS   = 7.6e-04
ah       = 8. * 2.0e+06
av       = 1.0e-03
kappah   = 8. * 1.0e+03
kappav   = 1.0e-04
```

Appendix B

Results for Bratu Problem

This appendix contains the extended data of the results of step size control and the corrector methods on the Bratu problem, that is discretized on a 31×31 grid. The corrector methods (for description see chapter 4) are tested for several initial step sizes. In the tables below we use the abreviations of Table 5.1.

init.	Newton ($N_{opt} = 4$)				
step	step	fail	ILUs	sols	time
0.7	20	4	78	78	10.9
0.8	23	5	97	97	13.7
0.9	22	5	73	73	10.4
1.0	25	6	92	92	13.0
1.1	23	5	95	95	13.7
1.2	23	5	74	74	10.5
1.3	26	7	95	95	13.5
av.	23	5	86	86	12.2

init.	Newton-chord ($N_{opt} = 7$)				
step	step	fail	ILUs	sols	time
0.7	23	7	23	129	6.4
0.8	28	10	28	139	7.6
0.9	20	5	20	105	5.8
1.0	22	7	22	129	6.4
1.1	22	7	22	128	6.4
1.2	24	8	24	121	6.3
1.3	24	8	24	125	6.5
av.	23	7	23	125	6.5

init.	ad. Shamanskii ($N_{opt} = 7$)				
step	step	fail	ILUs	sols	time
0.7	24	8	42	60	6.7
0.8	24	8	30	65	5.6
0.9	22	7	25	60	4.9
1.0	27	10	30	67	5.8
1.1	27	10	31	67	6.1
1.2	25	9	28	64	5.4
1.3	24	8	29	67	5.8
av.	25	9	31	64	5.8

init.	var. ad. Shamanskii ($N_{opt} = 7$)				
step	step	fail	ILUs	sols	time
0.7	25	7	23	91	6.4
0.8	26	8	26	87	6.3
0.9	25	7	27	94	6.7
1.0	28	10	23	87	6.0
1.1	21	6	18	76	4.8
1.2	26	8	32	108	7.7
1.3	25	8	28	92	6.7
av.	25	8	25	91	6.4

Appendix C

Results for the Ocean Circulation Problem

This appendix contains the extended data of the results of step size control and the corrector methods on the ocean circulation problem. The corrector methods (for a description see Chapter 4) are tested for several initial step sizes. In the tables below we use the abreviations of Table 5.1.

init.	Newton ($N_{opt} = 4$)				
step	step	fail	ILUs	sols	time
0.3	23	0	85	85	1238
0.5	28	2	94	94	1758
0.7	22	1	83	83	1336
0.9	28	2	101	101	1618
1.1	31	3	108	108	1981
av.	23	2	94	94	1586

init.	Newton-chord ($N_{opt} = 7$)				
step	step	fail	ILUs	sols	time
0.3	33	4	33	178	700
0.5	35	5	35	185	783
0.7	34	5	34	179	741
0.9	35	5	35	188	796
1.1	37	7	37	214	891
av.	35	5	35	189	782

init.	ad. Shamanskii ($N_{opt} = 7$)				
step	step	fail	ILUs	sols	time
0.3	30	4	39	147	786
0.5	28	4	36	141	703
0.7	29	4	40	151	787
0.9	27	3	38	153	788
1.1	29	4	36	153	761
av.	29	4	38	149	765

Appendix D

The Bifurcation in the Ocean Circulation Problem

This appendix gives some insight in the bifurcation point of the ocean circulation problem. To illustrate what happens at the bifurcation, we use the same example as in chapter 5: the continuation process for the temperature coefficient η_T for a box in the Atlantic ocean. We use the adaptive Shamanskii method ($N_{opt} = 7$) and take ananinitial step size 0.7.
During the continuation process we save two solutions on the branch, one before and one after the pitchfork bifurcation. In Figure D these two points are circeled.
We show some properties of the solution in each point by a nu,ber of graphs in Figure D.2. We plot the temperature profile in the middle meridian of the box, the meridional overturning (the velocity vertically averaged over the box) and the velocity field at the surface. In the graphs of the meridional overturning dotted lines denote a flow into the paper, while solid lines denote a flow out of the paper.

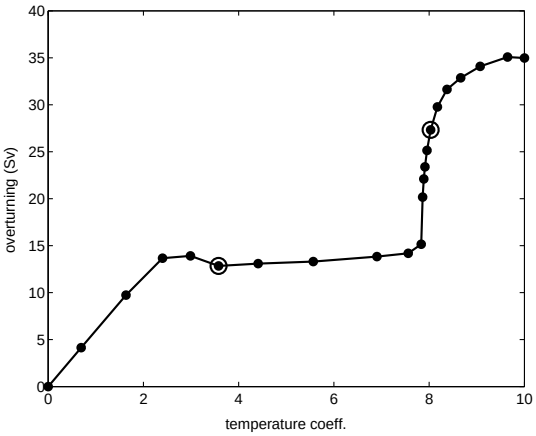

Figure D.1: The solution is shown at the encircled points.

In the graphs we see that the quality of the flow changes at the bifurcation point. Instead of a flow rotating in a single direction, we get two flows rotating at oposite directions, one at the surface and one at the bottom of the box. The continuation parameter η_T controls the amplitude of the atmospheric temperature function. A change in this parameter is expected to have some influence on the temperature profile. In the first plots in Figure D.2 we see that after the bifurcation, the extremes in the temperature are larger, than before the bifurcation. At the bottom the water is colder and at the surface it is hotter.

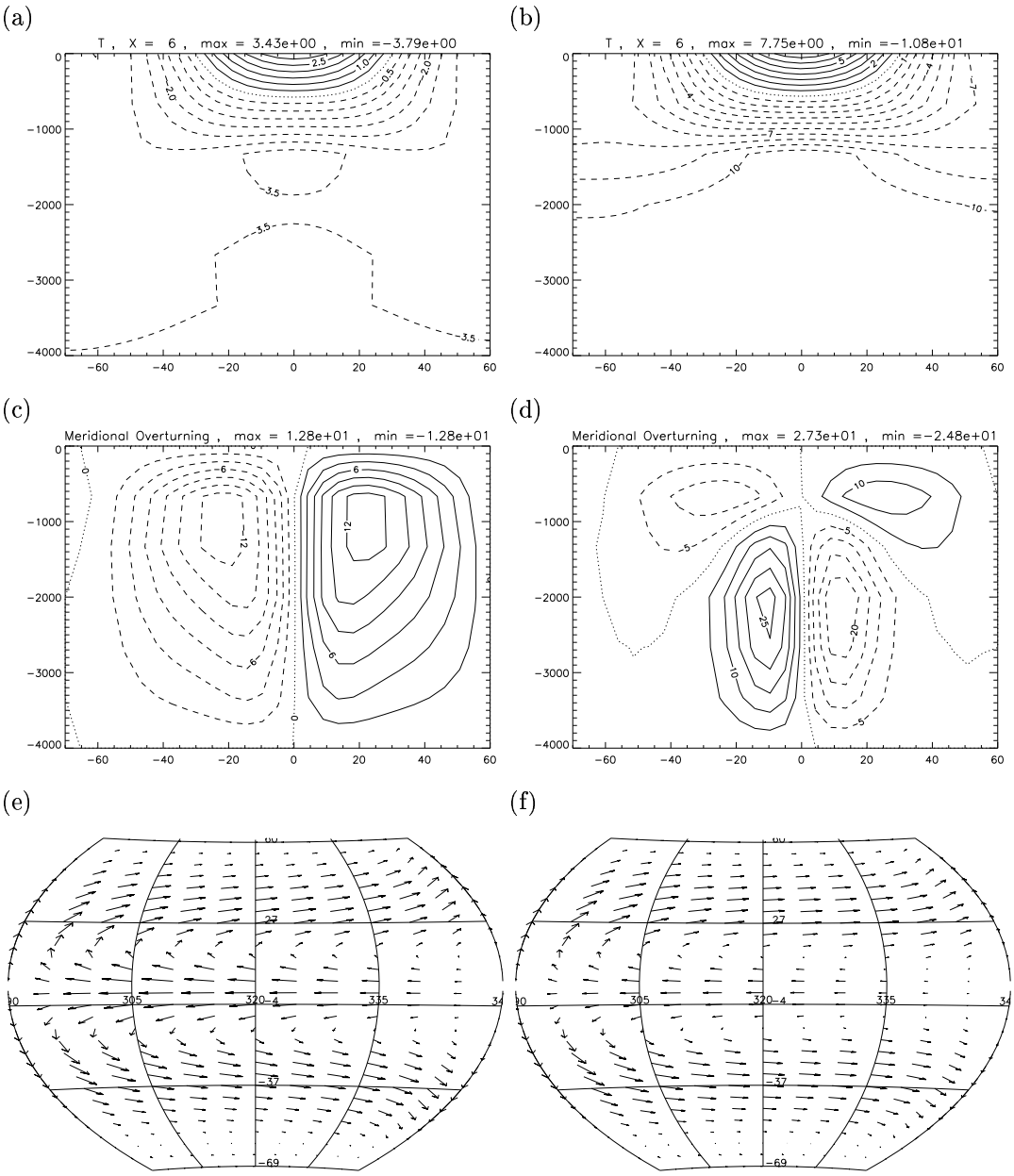

Figure D.2: The Temperature profile in the middle meridian of the box (a,b), the meridional overturning (c,d) and the velocity field at the surface (e,f) for the point before the bifurcation (left figures, a,c,e) and the point after the bifurcation (right figures, b,d,f).

Appendix E

Example of Construction of ILU Based on Nested-Dissection

We have written a program in MATLAB for the construction of the ILU factorization based on a nested-dissection ordering. In this appendix we show an example of the output of the program for the Poisson equation.
The grid contains on each level `N1` cells in horizontal direction and `N2` cells in vertical direction, with `n1` points on the horizontal separators and `n2` points on the vertical separators. With each elimination of separators, the number of cells in horizontal or vertical direction is halved. For an illustration of this process see Figure 9.1.

```
drop tolerance 1.0000e-01

STEP III) constructing incomplete LU decomposition ...

[ n1 n2 N1 N2 ] =       128 128
at step   1
[ n1 n2 N1 N2 ] =     1 128 64
4096 of the 16641 unknowns eliminated exactly
at step   2
[ n1 n2 N1 N2 ] =  1  1 64 64
2048 of the 12545 unknowns eliminated exactly
at step   3
[ n1 n2 N1 N2 ] =  1  3 64 32
2112 of the 10497 unknowns eliminated approximately
3072 of the 8385 unknowns eliminated exactly
at step   4
[ n1 n2 N1 N2 ] =  1  3 32 32
1056 of the 5313 unknowns eliminated approximately
1056 of the 4257 unknowns eliminated approximately
1056 of the 3201 unknowns eliminated approximately
 512 of the 2145 unknowns eliminated exactly
at step   5
[ n1 n2 N1 N2 ] =  1  1 16 32
 544 of the 1633 unknowns eliminated approximately
 256 of the 1089 unknowns eliminated exactly
at step   6
[ n1 n2 N1 N2 ] =  1  1 16 16
 128 of the  833 unknowns eliminated exactly
at step   7
[ n1 n2 N1 N2 ] =  1  3 16  8
 144 of the  705 unknowns eliminated approximately
 192 of the  561 unknowns eliminated exactly
at step   8
[ n1 n2 N1 N2 ] =  1  3  8  8
  72 of the  369 unknowns eliminated approximately
  72 of the  297 unknowns eliminated approximately
  72 of the  225 unknowns eliminated approximately
  32 of the  153 unknowns eliminated exactly
at step   9
[ n1 n2 N1 N2 ] =  1  1  4  8
  40 of the  121 unknowns eliminated approximately
  16 of the   81 unknowns eliminated exactly
at step  10
[ n1 n2 N1 N2 ] =  1  1  4  4
  20 of the   65 unknowns eliminated approximately
   8 of the   45 unknowns eliminated exactly
at step  11
[ n1 n2 N1 N2 ] =  1  1  4  2
  12 of the   37 unknowns eliminated approximately
   4 of the   25 unknowns eliminated exactly
at step  12
[ n1 n2 N1 N2 ] =  1  1  2  2
   6 of the   21 unknowns eliminated approximately
   6 of the   15 unknowns eliminated approximately
at step  13
[ n1 n2 N1 N2 ] =     1  2  1
   3 of the    9 unknowns eliminated approximately
at step  14
[ n1 n2 N1 N2 ] =  1     1  1
   2 of the    6 unknowns eliminated approximately
after last elimination
[ n1 n2 N1 N2 ] =        1  1
time to construct LU+E decompostion: 1.934e+01

STEP III)  ... finished



STEP IV)  testing quality of ILU decomposition ...

number of dropped sets :    16
nonzeros A : 148225
nonzeros U + L: 425146
relative fill: 2.86825
gmres(30) converged at outer iteration 1
  (inner iteration 13)
  to a solution with relative residual 7.1e-07
difference with exact solution 1.38047e-05
time to solve system with pcg/gmres: 2.137e+01

STEP IV)  ... finished
```

Bibliography

[AG90] Eugene L. Allgower and Kurt Georg. *Numerical continuation methods.* Springer-Verlag, Berlin, 1990. An introduction.

[Bot98] E. F. F. Botta. Dictaat eindige differentie methoden, 1998.

[BP88] James H. Bramble and Joseph E. Pasciak. A preconditioning technique for indefinite systems resulting from mixed approximations of elliptic problems. *Math. Comp.*, 50(181):1–17, 1988.

[Bre97] Claude Brezinski. *Projection methods for systems of equations.* North-Holland Publishing Co., Amsterdam, 1997.

[BW99] E. F. F. Botta and F. W. Wubs. Matrix renumbering ILU: an effective algebraic multilevel ILU preconditioner for sparse matrices. *SIAM J. Matrix Anal. Appl.*, 20(4):1007–1026 (electronic), 1999. Sparse and structured matrices and their applications (Coeur d'Alene, ID, 1996).

[CDFT92] P. Chin, E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang. Preconditioned conjugate gradient methods for the incompressible Navier-Stokes equations. *Internat. J. Numer. Methods Fluids*, 15(3):273–295, 1992.

[DOWB01] Henk A. Dijkstra, Hakan Oksuzoglu, Fred Wubs, and Eugen F.F. Botta. A fully implicit model of the three-dimensional thermohaline ocean circulation. *J. Comp. Phys.*, 173:685–715, 2001.

[Elm01] Howard C. Elman. Preconditioning for saddle point problems arising in computational fluid dynamics. Technical Report 4311, Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, 2001.

[GV98] Gene H. Golub and Denis Vanderstraeten. On the preconditioning of matrices with a dominant skew-symmetric component. Technical Report SCCM-98-16, Stanford University, 1998.

[JMM+00] V. John, G. Matthies, T. I. Mitkova, L. Tobiska, and P. S. Vassilevski. A comparison of three solvers for the incompressible Navier-Stokes equations. In *Large-scale scientific computations of engineering and environmental problems, II (Sozopol, 1999)*, pages 215–222. Vieweg, Braunschweig, 2000.

[JT00] V. John and L. Tobiska. Numerical performance of smoothers in coupled multigrid methods for the parallel solution of the incompressible Navier-Stokes equations. *Internat. J. Numer. Methods Fluids*, 33:453–473, 2000.

[Lev01] R.C. Levine. Improvements of a numerical continuation code for ocean circulation problems. Master's thesis, Rijksuniversiteit Groningen, August 2001.

[Mav95] D.J. Mavriplis. Three-dimensional multigrid Reynolds-averaged Navier-Stokes solver for unstructured meshes. *AIAA Journal*, 33:445–453, 1995.

[Meu99] G. Meurant. *Computer solution of large linear systems.* North-Holland Publishing Co., Amsterdam, 1999.

[Ols99] Maxim A. Olshanskii. An iterative solver for the Oseen problem and numerical solution of incompressible Navier-Stokes equations. *Numer. Linear Algebra Appl.*, 6(5):353–378, 1999.

[OR01] Maxim A. Olshanskii and Arnold Reusken. Grad-div stabilization for Stokes equations. Technical Report 208, Institut für Geometrie und Praktische Mathematik, 2001.

[PK94] Vijayan Parthasarathy and Y. Kallinderis. New multigrid approach for three-dimensional unstructured, adaptive grids. *AIAA Journal*, 32(5):956–963, 1994.

[Sey94] Rüdiger Seydel. *Practical bifurcation and stability analysis.* Springer-Verlag, New York, second edition, 1994. From equilibrium to chaos.

[Stü99] K. Stüben. Algebraic multigrid (amg): An introduction with applications. Technical report, GMD - Forschungszentrum Infromationstechnik GmbH, 1999.

[Van86] S. P. Vanka. Block-implicit multigrid solution of Navier-Stokes equations in primitive variables. *J. Comput. Phys.*, 65(1):138–158, 1986.

[Vel94] A. E. P. Veldman. Dictaat numerieke stromingsleer, 1994.

[Wil96] S. Ø. Wille. A non-linear adaptive tri-tree multigrid solver for finite element formulations of the Navier-Stokes equations. *Internat. J. Numer. Methods Fluids*, 22(11):1041–1059, 1996.

[Wit90] Gabriel Wittum. On the convergence of multi-grid methods with transforming smoothers. Theory with applications to the Navier-Stokes equations. *Numer. Math.*, 57(1):15–38, 1990.

[Zul02] Walter Zulehner. Analysis of iterative methods for saddle point problems: a unified approach. *Math. Comp.*, 71(238):479–505 (electronic), 2002.