Efficient solution techniques for the incompressible Stokes problem

Joost van Zwieten

May 28, 2010

Contents

Chapter 1

Introduction

Culgi — pronounce as 'tsjulgi' — is a company specialised in computational chemistry. The company develops software, also called Culgi, for the modelling of soft matter, e.g. liquids, colloids and polymers. Models which distinguishes single atoms of molecules and mutual force between atoms are used to investigate the structure of molecules. On a slightly coarser scale the dynamics of single molecules or a small cluster are simulated, for instance to study the interaction between different types of molecules or the structure in steady state. On a mesoscopic scale molecules are modelled as a continuum, which allows computation on larger domains at the cost of a less accurate model. The software is used, among others, by pharmacuetical and petroleum industries, for example as an aid for laboratory research.

One of the models incorporates the steady incompressible Stokes equations for the computation of a velocity field. The velocity field is used in a time integration method and computed every time step. Currently, the linear Stokes equations are solved using a non-linear equation solver. The purpose of this report is to give an overview of efficient solution techniques for the Stokes equations. Furthermore, the performance of the current implementation is investigated and compared with a Krylov subspace method.

The report is divided into five parts. Chapter 2 presents a brief derivation of the Stokes equations out of the Navier-Stokes equations and defines the boundary conditions used for the molecular modelling. Chapter 3 is an overview of solution techniques for general matrices, focussed on Krylov subspace methods. Chapter 4 presents several solution techniques based on subsystems of the Stokes equations. In Chapter 5 test problems are defined and the current implementation of the Stokes solver is investigated. In Chapter 6 the future research is formulated.

Chapter 2

Problem definition

In Section 2.1 the a model for molecular dynamics is presented. Section 2.2 describes the Navier-Stokes equations. In Section 2.3 the Stokes equations are derived from the Navier-Stokes equations. In Section 2.4 the Stokes equations are discretised using finite differences on a staggered grid. Finaly, Section 2.5 defines the specific boundary conditions for the molecular dynamics.

notation

2.1 Molecular dynamics

One of the models implemented in Culgi describes the dynamic behaviour of a mixture of molecules using a density field. Each molecule type i is assigned a density field ρ_i. The total density of the mixture $(\rho_i)_i)$ is constant. Every molecule type has a certain viscosity ν_i. The local viscosity of the fluid is approximated by the sum of the viscosities ν_i weighted by the local densities ρ_i, $\nu = \nu_i \rho_i$.

The dynamics of the molecules, or actually densities, are computed by an *evolution equation* which uses a for instance a convection diffusion process. The result is a density field per molecule type which is variable in time, $\rho_i(\vec{x}, t)$. The evolution equation requires a velocity which is computed by the incompressible Stokes equations, discussed below.

2.2 The Navier-Stokes equations

The dynamics of fluids can be described by classical conservation laws, in particular conservation of mass, momentum and energy. While fluids consist of particles — anything from small molecules to large polymers — they are modelled as a continuous medium, i.e. quantities such as velocity and density are everywhere defined. This assumption is called the continuum hypothesis.

The conservation of mass, written as a partial differential equation, is given by

$$\frac{\partial \rho}{\partial t} + (\rho u_\alpha)_{,\alpha} = 0, \tag{2.1}$$

where ρ is the local density of the fluid and u_α the velocity in direction α. This equation is also called the continuity equation.

A flow is incompressible if the density (in the neighbourhood) of each particle does not change in time. Put differently, as particles travel, the accompanying density travels along. The continuity equation for incompressible flow is given by

$$u_{\alpha,\alpha} = 0. \tag{2.2}$$

Note that incompressible flow does not imply that the density is constant in space.

The conservation of momentum is given by

$$\frac{\partial \rho u_\alpha}{\partial t} + (\rho u_\alpha u_\beta)_{,\beta} = -p_{,\alpha} + 2\left(\nu\left(e_{\alpha\beta} - \frac{1}{3}\Delta\delta_{\alpha\beta}\right)\right)_{,\beta} + \rho f^b_\alpha. \tag{2.3}$$

where p is the pressure, ν the viscosity, $\vec{f^b}$ is the body force — force acting on a particle —, $e_{\alpha\beta} = \frac{1}{2}(u_{\alpha,\beta} + u_{\beta,\alpha})$ the rate of strain tensor and $\Delta = e_{\alpha\alpha}$. The incompressible form is obtained by applying the incompressible continuity equation (2.2),

$$\rho\frac{\partial u_\alpha}{\partial t} + \rho u_{\alpha,\beta} u_\beta = -p_{,\alpha} + \left(\nu\left(u_{\alpha,\beta} + u_{\beta,\alpha}\right)\right)_{,\beta} + \rho f^b_\alpha. \tag{2.4}$$

The steady incompressible form of the momentum equation

$$\rho u_{\alpha,\beta} u_\beta = -p_{,\alpha} + \left(\nu\left(u_{\alpha,\beta} + u_{\beta,\alpha}\right)\right)_{,\beta} + \rho f^b_\alpha. \tag{2.5}$$

2.3 The Stokes equations

In case of viscous flow, the inertia term in the momentum equation may be neglected. For the steady incompressible form (2.5) this simplification leads to the following linear equation.

$$\left(\nu\left(u_{\alpha,\beta} + u_{\beta,\alpha}\right)\right)_{,\beta} - p_{,\alpha} = -\rho f^b_\alpha. \tag{2.6}$$

Together with the incompressible continuity equation (2.2), these equations are called the Stokes equations.

It is convenient to reformulate the Stokes equations into a dimensionless form. For this, the position $\vec{x}$, velocity $\vec{u}$ and the density ρ are made dimensionless as follows,

$$\vec{x}' = \frac{\vec{x}}{\vec{x}_0}, \qquad \vec{u}' = \frac{\vec{u}}{u_0}, \qquad \rho' = \frac{\rho}{\rho_0}. \tag{2.7}$$

Applying these quantities to the Stokes equations (2.6) and division by $\rho_0 u_0 x_0$ yields

$$\frac{1}{\rho_0 u_0 x_0}\left(\nu\left(u'_{\alpha,\beta}+u'_{\beta,\alpha}\right)\right)_{,\beta}-\frac{x_0}{u_0^2\rho_0}p_{,\alpha}=-\frac{x_0}{u_0^2}\rho' f_\alpha^{b\prime}, \tag{2.8}$$

where (spatial) derivatives are now with respect to the dimensionless $\vec{x}'$. Leaves the transformation of the pressure p, the bodyforce $\vec{f}^b$ and the viscosity ν,

$$p'=\frac{px_0}{u_0^2\rho_0}, \qquad \vec{f}^{b\prime}=\frac{\vec{f}^b x_0}{u_0^2}, \qquad \nu'=\frac{\nu}{\rho_0 u_0 x_0}. \tag{2.9}$$

The dimensionless Stokes equations are given by

$$\left(\nu'\left(u'_{\alpha,\beta}+u'_{\beta,\alpha}\right)\right)_{,\beta}-p'_{,\alpha}=-\rho' f_\alpha^{b\prime}. \tag{2.10}$$

By assumption the bodyforce acts only on molecules of the same type, hence for every molecule type i there is a $\vec{f}_i^b$. The Stokes equations are slightly modified to incorporate the different forces as follows,

$$\left(\nu'\left(u'_{\alpha,\beta}+u'_{\beta,\alpha}\right)\right)_{,\beta}-p'_{,\alpha}=-\rho'_i f_{i\alpha}^{b\ \prime}, \tag{2.11}$$

where ρ_i is the density of molecule type i. Note that the density of all molecules together is simply the sum of the densities of molecule types, $\rho=(\rho_i)_i$.

2.4 Discretisation

The Stokes equations (2.6) are discretised using finite difference methods on an equidistant staggered grid. The domain is partitioned into cubes, all of the same size. The pressure and viscosity nodes are located at the center of the cubes. The velocity nodes are centered at the faces of the cubes, such that the direction of the velocity coincides with the normal of the face, e.g. the x-velocity node is located at the center of the plane facing the x-direction. Figure 2.1 shows a two dimensional representation of the staggered grid.

Let $\tilde{u}_i^\alpha, \tilde{p}_j, \tilde{\nu}_j$ represent the discrete counterparts of u_α, p, ν at grid points i and j. The next node (of the same type) in direction α is denoted by $i+\alpha$. For example, let α be the x-direction. Then $\tilde{u}_{i+\alpha}^\alpha$ is the velocity node to the right of $\tilde{u}_i^\alpha$ and $\tilde{p}_{i+\alpha/2}$ is the pressure in between. See also Figure 2.2.

Let i be a grid point centered at a velocity node $\tilde{u}^\alpha$. The following equation is an approximation for (2.6)

$$\mathcal{N}\left(\left(\nu u_{\alpha,\beta}\right)_{,\beta}\right)_i+\mathcal{N}\left(\left(\nu u_{\alpha,\beta}\right)_{,\beta}\right)_i-\mathcal{N}\left(p_{,\alpha}\right)_i=-\left(\tilde{f}^b\right)_i^\alpha, \tag{2.12}$$

where $\mathcal{N}(\cdot)_i$ stands for the numerical approximation of $\cdot$ at position i. The derivative of the pressure at i in direction α is approximated by the first order central approximation around i:

$$\mathcal{N}\left(p_{,\alpha}\right)_i=h^{-1}\left(\tilde{p}_{i+\frac{\alpha}{2}}-\tilde{p}_{i-\frac{\alpha}{2}}\right), \tag{2.13}$$

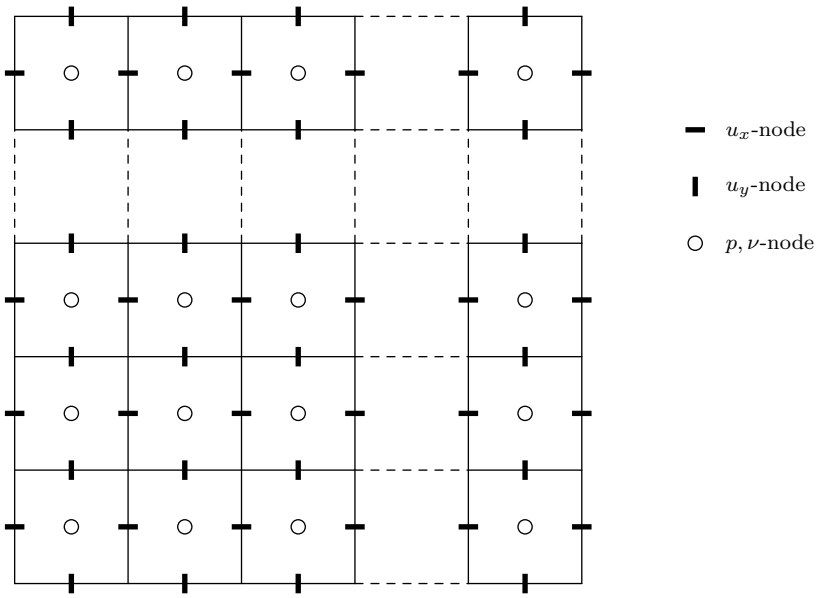

Figure 2.1: Slice of the three dimensional staggered grid.

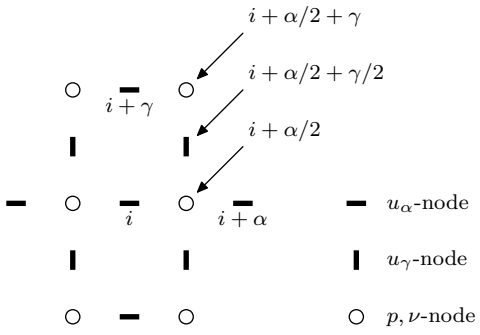

Figure 2.2: Stencil centered around a u_α-node, $\alpha \neq \gamma$.

where h is the (dimensionless) grid size.

$$\mathcal{N}\left(\left(\nu u_{\alpha,\beta}\right)_{,\beta}\right)_i = h^{-1}\sum_\beta \left(\tilde{\nu}_{i+\beta/2}\mathcal{N}\left(u_{\alpha,\beta}\right)_{i+\beta/2} - \tilde{\nu}_{i-\beta/2}\mathcal{N}\left(u_{\alpha,\beta}\right)_{i-\beta/2}\right). \tag{2.14}$$

The derivatives $u_{\alpha,\beta}$ are again approximated by a first order central scheme:

$$\mathcal{N}\left(u_{\alpha,\beta}\right)_{i+\beta/2} = h^{-1}\left(\tilde{u}^\alpha_{i+\beta} - \tilde{u}^\alpha_i\right). \tag{2.15}$$

$$\mathcal{N}\left(\left(\nu u_{\beta,\alpha}\right)_{,\beta}\right)_i = h^{-1}\sum_\beta \left(\tilde{\nu}_{i+\beta/2}\mathcal{N}\left(u_{\beta,\alpha}\right)_{i+\beta/2} - \tilde{\nu}_{i-\beta/2}\mathcal{N}\left(u_{\beta,\alpha}\right)_{i-\beta/2}\right), \tag{2.16}$$

where the derivatives $u_{\beta,\alpha}$ are approximated by a first order central scheme:

$$\mathcal{N}\left(u_{\beta,\alpha}\right)_{i+\beta/2} = h^{-1}\left(\tilde{u}^\beta_{i+\beta/2+\alpha/2} - \tilde{u}^\beta_{i+\beta/2-\alpha/2}\right). \tag{2.17}$$

Note that the viscosity at $i+\beta/2$ is not available if $\beta \neq \alpha$. Four point average:

$$\tilde{\nu}_{i+\beta/2} = \frac{\tilde{\nu}_{i+\alpha/2} + \tilde{\nu}_{i-\alpha/2} + \tilde{\nu}_{i+\alpha/2+\beta} + \tilde{\nu}_{i-\alpha/2+\beta}}{4} \tag{2.18}$$

2.5 Boundary conditions

One is often interested in the behaviour of molecules, the miscibility of different molecules and the phases which result from the demixing of molecules, on a scale comparable to the size of the molecules. Simulating, for instance, a whole reactor where polymers are created is infeasible, simply because a grid with the necessary detail would be enormous. Therefore, only a small portion of the whole domain is simulated. Since this subdomain lacks real boundaries it is natural to assume that the fluid is periodic in the neighbourhood of the subdomain.

Let Ω be the three dimensional (sub)domain under investigation, $\Omega = [0, s_1]\times[0, s_2]\times[0, s_3]$, where $\vec{s}$ is the (dimensionless) size of the domain. The periodicity written in condensed form,

$$\vec{u}(\vec{x} + \vec{n}\vec{s}) = \vec{u}(\vec{x}), \qquad p(\vec{x} + \vec{n}\vec{s}) = p(\vec{x}), \tag{2.19}$$

where $n_\alpha \in \{0, 1\}$, hence $(\vec{n}\vec{s})_\alpha \in \{0, s_\alpha\}$ corresponds to no translation or a translation in direction α by s_α.

Certain phenomena, such as stirring in a mixture or the behaviour of a flow near a solid wall, may be approximated by a imposing a velocity difference between two opposite sides of Ω. Due to the periodicity this is equivalent to imposing a velocity profile onto the whole domain in the following way. Let $\vec{u}^{\text{shear}}$ be the velocity difference between the sides $x_\alpha = 0$ and $x_\alpha = s_\alpha$. The shear velocity in direction α should equal zero, $u^{\text{shear}}_\alpha = 0$. The periodicity for the velocity becomes

$$\vec{u}(\vec{x} + \vec{n}\vec{s}) = \vec{u}(\vec{x}) + n_\gamma\delta_{\alpha\gamma}\vec{u}^{\text{shear}}(\vec{x}), \tag{2.20}$$

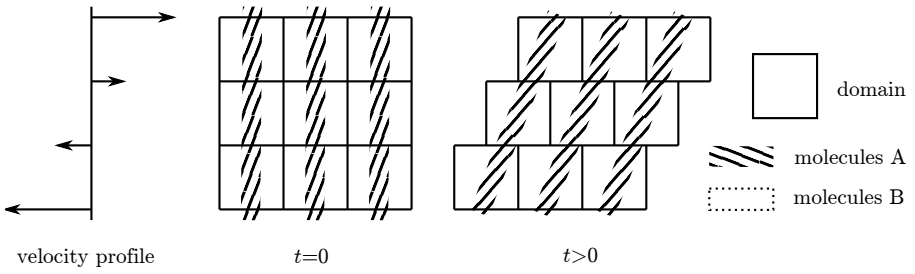

Figure 2.3: Foo

where $\delta_{\alpha\gamma}$ is one if $\alpha = \gamma$ and zero otherwise.

The evolution equation computes the distribution of molecules after some time t under influence of the velocity computed by the Stokes equations, in which the shear velocity is incorporated. The time integration influences the periodicity in a way which is perhaps best explained with an illustration. Figure 2.3 shows a square domain filled with two types of molecules, A and B. The molecules repel each other to some extent and form a so called laminar phase. The evolution equation computes the densities of the different molecules at some time $t > 0$ given the velocity profile shown in the figure. The vertical phases are slanted a little. From this follows that the periodicity is in some sense shifted.

Pouring this phenomenon into a mathematical equation yields the following periodicity for the velocity

$$\vec{u}\left(\vec{x} + \vec{n}\vec{s} + n_\alpha t\vec{u}^{\text{shear}}\right) = \vec{u}(\vec{x}) + n_\gamma \delta_{\alpha\gamma}\vec{u}^{\text{shear}}(\vec{x}), \tag{2.21}$$

and for the pressure

$$p\left(\vec{x} + \vec{n}\vec{s} + n_\alpha t\vec{u}^{\text{shear}}\right) = p(\vec{x}). \tag{2.22}$$

Chapter 3

Solution techniques for general matrices

In this chapter various techniques are described for solving

$$Ax = b \tag{3.1}$$

where A is a real $n \times n$-matrix and x, b vectors in $\mathbb{R}^n$.

In the most crude sense, there are two types of methods for solving a sparse linear system $Ax = b$. First, one could use a direct method, adapted to benefit from the sparsity of A.

Secondly, there are iterative methods, which generate a sequence of approximate solutions $(x_k)_k$ to $Ax = b$. Iterates are usually obtained by updating the previous vector with a matrix-vector product involving A or an approximation of A.

Iterative methods have several advantages over direct methods. While direct methods give the exact solution of a system — at least using exact arithmetic — iterative methods enable one to obtain an approximate solution with a controllable accuracy. Besides, iterative methods can benefit from an initial guess. This is, for example, very useful when a time dependent system is solved with relatively small time derivatives. And iterative methods are in general more suited for sparse matrices.

Section 3.1 describes basic iterative methods. The more advanced Krylov subspace methods are described in Section 3.2. Section 3.3 presents a brief introduction into preconditioning. The last section describes the deflation method.

3.1 Basic iterative methods

A large class of iterative methods is based on solving a system using a (good) approximation of matrix A, say M, and a right hand side wich corrects for

the difference between A and M. To be more precise, matrix A is split into a nonsingular matrix M and a matrix N, $A = M - N$, and the system

$$Mx_{k+1} = Nx_k + b, \tag{3.2}$$

is solved iteratively. This method is consistent with the original system. Indeed, if x solves $Ax = b$, then $Mx = Nx + Ax = Nx + b$. However, not every splitting will lead to a convergent process independent of the initial guess.

To investigate the convergence, consider the iteration matrix $G = M^{-1}N = I - M^{-1}A$. Substituting G for M and N in (3.2) gives

$$x_{k+1} = Gx_k + f, \tag{3.3}$$

where $f = M^{-1}b$. The absolute error after k iterations is given by

$$x_k - A^{-1}b = Gx_{k-1} + f - A^{-1}b = G(x_{k-1} - A^{-1}b) = G^k(x_0 - A^{-1}b). \tag{3.4}$$

From this follows that the iteration converges for any initial guess and right hand side if and only if the spectral radius of G is less than one, or equivalently, if all eigenvalues of G lie *inside* the unit circle.

This result leads to the question: what is a good splitting? The total computational time is determined by the number of iterations, and as such the required accuracy, and the amount of work per iteration. As stated above, the rate of convergence is proportional to the spectral radius of $G = M^{-1}N$, which, in turn, is a measure for equality of A and M. As an extreme example — not really iterative though — consider $A = M$; then $G = 0$ and the method converges in a single iteration.

The amount of work per iteration is in most cases dominated by solving a system involving matrix M. Setting M equal to the identity matrix yields Richardson's iteration, discussed in the next section.

3.1.1 Richardson's iteration

One of the simplest iterative schemes for solving $Ax - b$ is Richardson's iteration, given by

$$x_{k+1} = x_k + \alpha_k(b - Ax_k), \tag{3.5}$$

where α_k is a possibly constant scalar. This corresponds to a splitting of $A = M_k - N_k$ with $M_k = I/\alpha_k$ and $N_k = I/\alpha_k - A$.

The coefficients α_k determine the rate of convergence. For simplicity, assume all α_k are constant for every iteration, $\alpha_1 = \alpha_2 = \cdots = \alpha_k$.

Convergence for this method is guaranteed for any initial guess x_0 and any right hand side b if the modules of all eigenvalues of the iteration matrix $G = I - \alpha A$ are less than one (Section 3.1), or equivalently,

$$\left|\frac{1}{\alpha} - \lambda\alpha\right| < \left|\frac{1}{\alpha}\right| \quad \forall\lambda \in \sigma(A). \tag{3.6}$$

From this follows immediately that all eigenvalues should be located in either the (strict) positive or the (strict) negative half plane. When matrix A is symmetric, this amounts to A being definite.

The optimal choice for α, without regard of initial guess and right hand side, is found by minimising the general convergence factor, which is equal to the spectral radius of G (Section 3.1). In case A is symmetric and, as required for convergence, definite, then all eigenvalues are real and the optimal value for $1/\alpha$ lies in the middle of the biggest and smallest eigenvalue of A (Saad 1996),

$$\alpha_{\text{opt}} = \arg\min_{\alpha} \rho(I - \alpha A) = \frac{2}{\lambda_{\max} + \lambda_{\min}}. \tag{3.7}$$

Problems: Rate of convergence can be very slow if there are both small and large eigenvalues. Estimating eigenvalues may be expensive, but is required for a good aproximation of the optimal α. Sensitivity of α near optimal value for large eigenvalues.

3.1.2 Steepest descent

The determination of the coeffient(s) α in Richardson's iteration is troublesome due to the need for possibly accurate estimations of eigenvalues. There is, however, a more elegant way for positive symmetric definite matrices which calculates an α_k — the variable is nonconstant per iteration — using a single matrix-vector multiplication.

The method of steepest descent is based on the minimisation of the function

$$\phi(x_*) = \|x_* - A^{-1}b\|_A^2 = (x_* - A^{-1}b)^T A(x_* - A^{-1}b), \tag{3.8}$$

of which the minimum solves the system $Ax = b$. By setting the gradient of ϕ equal to zero, one can see that this is indeed true. The minimisation is done iteratively. Assume k iterations are performed and the current approximate solution is x_k. At this point the gradient of ϕ is determined, which can be viewed as the (negative) direction of steepest descent at x_k. The next iterate is formed by traveling a certain distance, say α_k, in the direction of steepest descent,

$$x_{k+1} = x_k - \alpha_k \nabla\phi(x_k) = x_k + \alpha_k(b - Ax_k). \tag{3.9}$$

Upto now, this is simply Richardson's iteration with nonconstant α_k. The distances α_k are chosen such that ϕ is minimised along the line from x_k towards steepest descent,

$$\alpha_k = \frac{\langle r_k, r_k \rangle_{l^2}}{\langle Ar_k, r_k \rangle_{l^2}}. \tag{3.10}$$

The absolute error of iteration $k > 0$ is bounded by (Saad 1996)

$$\|d_k\|_A = \|A^{-1}b - x_k\|_A \leq \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} \|d_{k-1}\|_A = g\|d_{k-1}\|_A, \tag{3.11}$$

where $\|\cdot\|_A$ is the A-norm defined by $\|y\|_A^2 = y^T Ay$. Since A is symmetric positive definite, all eigenvalues are positive, hence $g < 1$ and d_k is strictly

smaller than the previous error. Applying the equation subsequently proves convergence,

$$\|d_k\|_A \leq g^k \|d_0\|_A \to 0 \quad \text{as} \quad k \to \infty. \tag{3.12}$$

However, if the spectral condition number $\kappa = \lambda_{\max}/\lambda_{\min}$ is large, then the rate of convergence

$$g = \frac{\kappa - 1}{\kappa + 1} \tag{3.13}$$

is close to one and convergence may be very slow.

3.2 Krylov subspace methods

An important class of iterative solution techniques is based on the idea that the solution of the problem $Ax = b$ can be approximated by a polynomial of A,

$$A^{-1}b \approx p(A)b. \tag{3.14}$$

If there is a good initial guess available, then the equivalent system $A(x - x_0) = b - Ax_0 = r_0$ can be solved. Usually, the approximation is obtained by means of an iterative process. Starting with a polynomial of zero degree, every iteration a degree is added to this polynomial to increase the accuracy. Doing so, the approximate solution after m iterations is given by

$$x_m = x_0 + p_{m-1}(A)r_0, \tag{3.15}$$

where p_{m-1} is a polynomial of degree $m-1$ or less. The generated approximate solutions are contained in the affine subspace $x_0 + \mathcal{K}_m(A, r_0)$, where $K_m(A, r_0)$ is the Krylov subspace defined by

$$\mathcal{K}_m(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}. \tag{3.16}$$

When there is no ambiguity, $\mathcal{K}_m(A, r_0)$ is simply denoted by $\mathcal{K}_m$.

The search space for the exact solution is $\mathbb{R}^n$. It is possible that the Krylov subspace can not become this big, which poses a problem when the exact solution is not contained in any Krylov subspace. However, it will be proved in the next section that this cannot happen. For the moment assume the dimension of the m-th Krylov subspace $\mathcal{K}_m$ is m, and the vectors $v_1, \ldots, v_m$ form a basis of $\mathcal{K}_m$. Let V_m be the $n \times m$-matrix whose columns are precisely these basis vectors. The approximate solution x_m can be written in terms of a vector $y_m \in \mathbb{R}^m$ as

$$x_m = x_0 + V_m y_m, \tag{3.17}$$

and the residual $b - Ax_m$ as

$$r_m = r_0 - AV_m y_m. \tag{3.18}$$

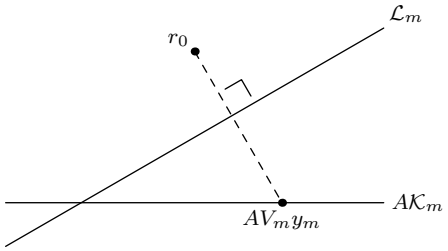

Figure 3.1: Illustration of a projection method onto the Krylov subspace: the initial residual r_0 is projected onto the Krylov subspace $\mathcal{K}_m$ orthogonal to the subspace of constraints $\mathcal{L}_m$.

3.2.1 Projection method

This leaves the question how to choose the approximate solution x_m or, equivalently, vector y_m. In a projection method the approximate solution is determined by constraining the residual r_m to be orthogonal to $\mathcal{L}_m \subset \mathbb{R}^n$, the subspace of constraints,

$$r_m = r_0 - AV_m y_m \perp \mathcal{L}_m. \tag{3.19}$$

In order to obtain a unique y_m, $\mathcal{L}_m$ should be of dimension m and such that every nonzero vector $f \in A\mathcal{K}_m$ is not orthogonal to $\mathcal{L}$. The result is a projection of the initial residual r_0 onto the subspace $A\mathcal{K}_m$ orthogonal to the subspace of constraints $\mathcal{L}_m$, illustrated in Figure 3.1.

The following theorem and corollary prove that a projection method onto the Krylov subspace eventualy finds the exact solution.

Theorem 3.1 (breakdown)**.** *If the Krylov subspace $\mathcal{K}_{m+1}$ is equal to $\mathcal{K}_m$, then the approximate solution $x_m \in \mathcal{K}_m$, obtained by a projection method, solves $Ax = b$, at least using exact arithmetic.*

Proof. Assume the m-th and $m+1$-th Krylov subspaces are equal, $\mathcal{K}_m = \mathcal{K}_{m+1}$. The approximate solution $x_m \in \mathcal{K}_m$ can be written as a linear combination of the vectors $r_0, Ar_0, \ldots, A^{m-1}r_0$. Premultiplying x_m by A yields a linear combination of $Ar_0, A^2r_0, \ldots, A^m r_0$, hence $Ax_m \in \mathcal{K}_{m+1} = \mathcal{K}_m$. But the residual $r_0 \in \mathcal{K}_m$ by construction. The constraint (3.19) implies $r_0 = Ax_m$ and $r_m = 0$. In other words, x_m solves the problem $Ax = b$. □

The following corollary is an immediate consequence.

Corollary 3.2 (exact solution)**.** *The approximate solution $x_n \in \mathcal{K}_n(A, r_0)$ obtained by a projection method is the exact solution of the problem $Ax = b$, where A is $n \times n$-dimensional, at least using exact arithmetic.*

Proof. This follows from Theorem 3.1 and the fact that every Krylov subspace is contained in $\mathbb{R}^n$. □

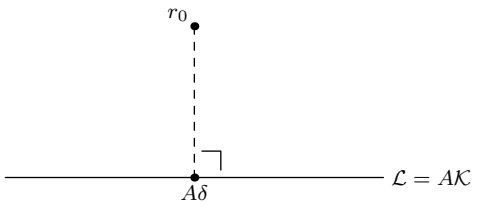

Figure 3.2: Illustration of the projection method when $\mathcal{L} = A\mathcal{K}$: $A\delta$ is the orthogonal projection of the initial residual r_0 onto $\mathcal{L}$.

3.2.2 Projection of the initial residual

In this section the special case $\mathcal{L}_m = A\mathcal{K}_m$ is investigated, where A is an arbitrary invertible matrix. A projection method constrains the residual r_m to be orthogonal to the subspace of constraints $\mathcal{L}_m$,

$$r_m = r_0 - AV_m y_m \perp A\mathcal{K}_m. \tag{3.20}$$

Since the columns of V_m span $\mathcal{K}_m$ and y_m searched in $\mathcal{R}^m$, the vector $AV_m y_m$ is essentially the orthogonal projection of r_0 into the subspace $A\mathcal{K}_m$. This is illustrated in Figure 3.2. Furthermore, the approximate solution is the result of the minimisation of residual two-norm over $y \in \mathbb{R}^m$,

$$y_m = \underset{y \in \mathbb{R}^m}{\arg\min} \|r_0 - AV_m y\|_2. \tag{3.21}$$

3.2.3 Projection of the initial error

In this section the special case $\mathcal{L} = \mathcal{K}$ in combination with a symmetric and positive definite matrix A is investigated. Let $d_0 = x - x_0$ be the initial error and $d_m = x - x_m$ the error of the approximate solution.

The projection process constrains the residual r_m to be orthogonal to $\mathcal{K}$. This is equivalent to

$$\langle r_0 - AV_m y_m, v_j \rangle = 0 \quad \forall 1 \leq j \leq m. \tag{3.22}$$

Since A is symmetric and positive definite, it defines an innerproduct denoted by $\langle \cdot, \cdot \rangle_A$. Substituting this innerproduct into (3.22) gives

$$\langle d_0 - V_m y_m, v_j \rangle_A = 0 \quad \forall i \leq j \leq m. \tag{3.23}$$

Vector $V_m y_m$ can be viewed as the A-orthogonal projection of the initial error d_0 onto subspace $\mathcal{K}_m$. Furthermore, this is equivalent to the minimisation

$$y_m = \underset{y \in \mathbb{R}^m}{\arg\min} \|d_0 - V_m y\|_A, \tag{3.24}$$

where $\| \cdot \|_A$ is the norm defined by $\|z\|_A = \sqrt{\langle z, z \rangle}$.

3.2.4 Full orthogonalisation method

The full orthogonalisation method is an orthogonal projection method onto the Krylov subspace K_m. It constructs an orthonormal basis for $\mathcal{K}_m$, which is used to transform the constraint (3.19) into a relatively easy to solve matrix vector problem involving an upper Hessenberg matrix.

Outline of the method

The objective is to determine an orthonormal basis $v_1, \ldots, v_m$ for the Krylov subspace $\mathcal{K}_m$, such that the first j vectors span $\mathcal{K}_j$, for all $j \leq m$. Note that this uniquely determines the vectors $v_1, \ldots, v_m$ and implies $v_1 = r_0/\|r_0\|_2$. Of course, one could simply apply a Gram-Schmidt orthogonalisation and normalisation to the defining vectors of the Krylov subspace, $r_0, Ar_0, \ldots, A^{m-1}r_0$, but it turns out to be more convenient to use the vectors $r_0, Av_1, Av_2, \ldots, Av_{m-1}$. This leads to the recurrence

$$h_{j+1,j}v_{j+1} = Av_j - \sum_{i=1}^{j} \frac{\langle Av_j, v_i\rangle}{\langle v_i, v_i\rangle} v_i = Av_j - \sum_{i=1}^{j} h_{ij}v_i, \tag{3.25}$$

where $h_{j+1,j}$ is chosen such that the resulting vector v_{j+1} is normal, $\|v_{j+1}\|_2 = 1$. This procedure is called Arnoldi's method. The following theorem proofs that the resulting set $\{v_1, \ldots, v_m\}$ indeed spans the Krylov subspace $\mathcal{K}_m$.

Theorem 3.3. *The vectors $v_1, \ldots, v_m$ obtained by setting $v_1 = r_0/\|r_0\|_2$ and applying (3.25) form an orthonormal basis of the Krylov subspace $\mathcal{K}_m$.*

Proof. The proof is by induction. The theorem clearly holds for $j = 1$. Assume the theorem holds for all integers $i \leq j$. Then $v_i \in \mathcal{K}_i$ and v_i can be written as $q_{i-1}(A)r_0$, where q_{i-1} is a polynomial of degree $i-1$. Observe that the next vector v_{j+1} becomes

$$h_{j+1,j}v_{j+1} = Aq_{j-1}(A)r_0 - \sum_{i=1}^{j} h_{ij}q_{i-1}(A)r_0, \tag{3.26}$$

hence v_{j+1} can be written as $q_j(A)r_0$ with q_j a polynomial of degree j. Besides, v_{j+1} is orthonormal to $\mathcal{K}_j$, which concludes the proof. □

Denote by V_m the $n \times m$-matrix whose columns are the vectors $v_1, \ldots, v_m$ and by H_m the $m \times m$ upper Hessenberg matrix consisting of the coefficients h_{ij},

$$H_m = \begin{bmatrix} h_{11} & h_{12} & \ldots & h_{1,m-1} & h_{1m} \\ h_{21} & h_{22} & \ldots & h_{1,m-1} & h_{2m} \\ & h_{32} & \ldots & h_{1,m-1} & h_{3m} \\ & & \ddots & \vdots & \vdots \\ & & & h_{m,m-1} & h_{mm} \end{bmatrix}. \tag{3.27}$$

The orthonormalisation (3.25) results in a system of equations using V_m and H_m,

$$AV_m = V_m H_m + h_{m+1,m}v_{m+1}e_m^T, \tag{3.28}$$

where e_m is the last canonical basis vector in $\mathbb{R}^m$. This equations is very useful for transforming the constraint (3.19) into a small and relatively easy to solve problem. First, recall that the initial residual r_0 is a scalar multiple of the first basis vector v_1, hence r_0 can be written as $V_m e_1 \|r_0\|_2$, where e_1 is the first canonical basis vector in $\mathbb{R}^m$. Applying this and equation (3.28) to the residual r_m yields

$$r_m = r_0 - AV_m y_m = V_m(\|r_0\|_2 e_1 - H_m y_m) - h_{m+1,m} v_{m+1} e_m^T y_m. \tag{3.29}$$

Since the subspace of constraints $\mathcal{L}_m = \mathcal{K}_m$ is spanned by the vectors $v_1, \ldots, v_m$, the constraint $r_m \perp \mathcal{L}_m$ (3.19) is equivalent to

$$H_m y_m = \|r_0\|_2 e_1. \tag{3.30}$$

If y solves this equation, then the residual follows easily from (3.29) and (3.30),

$$r_m = -h_{m+1,m}(e_m^T y) v_{m+1}. \tag{3.31}$$

Solving equation (3.30) is still not trivial. Fortunately, there is an easy way to transform the upper Hessenberg matrix H_m into a triangular matrix. Consider the first two rows of H_m,

$$H_m|_{[1,2]\times[1,m]} = \begin{bmatrix} h_{11} & h_{12} & \ldots & h_{1m} \\ h_{21} & h_{22} & \ldots & h_{2m} \end{bmatrix}. \tag{3.32}$$

Element $(1,2)$ can be nullified by premultiplying these rows with a 2×2 rotation matrix G_1,

$$G_1 \, H_m|_{[1,2]\times[1,m]} = \begin{bmatrix} h'_{11} & h'_{12} & \ldots & h'_{1m} \\ 0 & h'_{22} & \ldots & h'_{2m} \end{bmatrix} = H'|_{[1,2]\times[1,m]}, \tag{3.33}$$

thereby changing the first two rows, but leaving all other rows of H_m intact. This process is called Givens rotation. Subsequently applying Givens rotations to (3.30) yields a triangular system,

$$U_m y_m = \|r_0\|_2 Q_m e_1, \tag{3.34}$$

where Q_m represents all Givens rotations $G_1, \ldots, G_{m-1}$, and $U_m = Q_m H_m$ is the upper triangular matrix resulting from H_m. For details and the derivation of the correct Givens matrices to obtain a triangular system, see Golub and Van Loan (1996).

Analysis

The only possibility of breakdown is in Arnoldi's method, when a vector v_j is zero. Normalisation of the vector is not possible and the method stops. By Theorem 3.1, the approximate solution at this point turns out to be the exact solution, hence this is a form of lucky breakdown.

The orthonormalisation process uses all previously obtained basis vectors to generate a new one. For big systems and large Krylov subspaces this requires a large amount of memory and every iteration the amount of work increases.

There are several ways to address this problem. The method can be repeated several times with a 'small' Krylov subspace until accuracy is reached (Restarted FOM). Or one could apply incomplete orthogonalisation by truncating the orthonormal basis — all except the k last orthonormal vectors are thrown away — (Incomplete Orthogonalisation Process). Both methods require less memory than FOM, but lost the optimality property.

3.2.5 Generalised minimal residual method

The generalised minimal residual method is a projection method onto the Krylov subspace $\mathcal{K}_m$ orthogonal to $\mathcal{L}_m = A\mathcal{K}_m$. As in FOM, GMRES uses Arnoldi's method to construct an orthonormal basis of $\mathcal{K}_m$.

Outline of the method

GMRES also uses Arnoldi's method for the generation of an orthonormal basis of the Krylov subspace $\mathcal{K}_m$. Due to the choice of the subspace of constraints, the projection process is equivalent to the minimisation of the residual norm

$$y_m = \underset{y \in \mathbb{R}^m}{\arg\min} \|r_0 - AV_m y\|_2. \tag{3.35}$$

For this method it is more convenient to write (3.28) as

$$AV_m = V_{m+1}\bar{H}_m, \tag{3.36}$$

where $\bar{H}_m$ is defined by

$$\bar{H}_m = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{1,m-1} & h_{1m} \\ h_{21} & h_{22} & \dots & h_{1,m-1} & h_{2m} \\ & h_{32} & \dots & h_{1,m-1} & h_{3m} \\ & & \ddots & \vdots & \vdots \\ & & & h_{m,m-1} & h_{mm} \\ & & & & h_{m+1,m} \end{bmatrix}. \tag{3.37}$$

Using (3.36) the residual r_m can be written as

$$r_m = r_0 - AV_m y_m = r_0 - V_{m+1}\bar{H}_m y_m = V_{m+1}(\beta e_1 - \bar{H}_{m+1} y_m). \tag{3.38}$$

Finally, since the columns of V_{m+1} are orthonormal, the minimisation of the residual becomes

$$y_m = \underset{y \in \mathbb{R}^m}{\arg\min} \|\beta e_1 - \bar{H}_{m+1} y\|_2. \tag{3.39}$$

This minimisation is a least-squares problem with $m + 1$ equations.

Equation (3.39) can be transformed into a triangular system with m equations by applying Givens rotations to $\bar{H}_{m+1}$, discussed in Section 3.2.4. For details and implementations, see Saad (1996).

Analysis

As in FOM, the GMRES method may suffer from breakdown in Arnoldi's method. However, the breakdown leads to an exact solution, hence will not be a problem. The approximate solution obtained by GMRES is the result of minimising the residual in a certain space. This optimality property is useful for analysing the convergence of the method.

As a consequence of using Arnoldi's method, GMRES shares some problems with FOM. The amount of memory and computations increases each iteration. One of the remedies is restarting the method after m iterations again and again until convergence is achieved, denoted by GMRES(m). This allows easy control over the amount of memory and effort. However, the optimality property is lost and the convergence may become very slow since the super linear convergence starts only after a couple of iterations.

3.2.6 Generalised conjugate residual method

The generalised conjugate residual method (Eisenstat, Elman, and Schultz 1983), GCR for short, is a projection method on $\mathcal{K}_m$ orthogonal to $\mathcal{L}_m = A\mathcal{K}_m$, hence mathematicaly equivalent to GMRES. While FOM and GMRES construct an orthonormal basis of the Krylov subspace, GCR builds a basis $\{p_0, \ldots, p_{m-1}\}$ such that all Ap_i are orthogonal, i.e. all p_i are A^TA-orthogonal. These search directions p_i are conjugated residuals, i.e. r_i A^TA-orthogonalised to all previous search directions.

The computation of a new basis vector p_m involves the current residual r_m, all previously computed vectors $p_0, \ldots, p_{m-1}$ and their A multiples $Ap_0, \ldots, Ap_{m-1}$. Although the latter set of vectors can be computed on the fly from the former, the repeated computation of these matrix vector products is considered too expensive. Therefore, both sets of vectors are simply stored, which doubles the amount of memory compared to GMRES. Furthermore, the computational effort is slightly higher than GMRES as well, despite the storage of all Ap_i.

One could choose to drop the optimality property in favour of memory consumption by applying incomplete orthogonalisation. A trivial way to do this, is truncation of the basis vectors: only the last k computed basis vectors are stored and used for the orthogonalisation. Contrary to FOM and GMRES, this requires little change to the algorithm.

3.2.7 GMRES recursive

The GCR method builds an approximate solution in the Krylov subspace by the following recurrence,

$$x_{k+1} = x_k + \alpha_k p_k, \tag{3.40}$$

where the α_k's are chosen such that the residual,

$$r_{k+1} = r_k - \alpha_k A p_k, \tag{3.41}$$

is minimised in two-norm. Assume, contrary to/in contrast with GCR, the search vectors p_k are free to choose. Since the residual should be minimised, a good search direction is such that Ap_k approximates r_k. Indeed, choosing $p_k = A^{-1}r_k$ solves the problem immediately, but this merely shifts all effort into the computation of p_k.

The above idea has lead to *GMRES recursive* (Vorst and Vuik 1994), GMRESR for short. This method consists of two nested solvers. The main solver is basicly GCR, modified to allow arbitrary (to be conjugated) search direction u_k instead of r_k. If neccessary, one could choose to truncate GCR to limit the amount of memory and work.

The search directions u_k are obtained by approximating $Au = r_k$ with GMRES. It can be shown that the convergence of the GCR loop does not improve if the accuracy of u_k is higher than the required accuracy of the approximate solution. In addition, the number of iterations of GMRES loop is usually limited by some constant m. The optimal choice of this constant is a trade-off between memory and work.

3.2.8 Minimum residual method

Minimum residual method (Paige and Saunders 1975), MINRES for short, is a projection method onto the Krylov subspace $\mathcal{K}_m$ orthogonal to $\mathcal{L}_m = A\mathcal{K}_m$. Only for symmetric matrices A.

Arnoldi's method constructs an orthonormal basis $\{v_1, \ldots, v_m\}$ of the Krylov subspace $\mathcal{K}_m$ and a matrix H_m which reconstructs A_m out of $V_m = [v_1, \ldots, v_m]$. By construction H_m is upper Hessenberg. If matrix A is symmetric then so is $H_m = V_m^T A V_m$. But this implies that H_m is tridiagonal and equation (3.25) simplifies to a recurrence using only the last two obtained basis vectors. Arnoldi's method simplified for symmetric matrices is called Lanczos algorithm and matrix H_m is renamed to T_m,

$$T_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \beta_m & \alpha_m \end{bmatrix}. \tag{3.42}$$

Matrix T_m is easily transformed into a lower triangular matrix $\bar{L}_m$ by applying a series of Givens rotations $G_{i,i+1}$ to every pair of concurrent columns i and $i+1$,

$$\bar{L}_m = T_m G_{1,2} \cdots G_{m-1,m} = T_m Q_m^T, \tag{3.43}$$

where Q_m^T is the product of all rotations. Since all Givens rotations are orthogonal, equation (3.43) defines the LQ-decomposition of T_m.

A projection method requires the residual r_m to be orthogonal to the subspace of constraints $\mathcal{L}_m$. This is equivalent to

$$V_m^T A^2 V_m y_m = V_m^T A r_0. \tag{3.44}$$

Applying (3.28) of Arnoldi's method and (3.43) to the matrix on the left hand side gives

$$V_m^T A^2 V_m = \bar{L}_m \bar{L}_m^T + \beta_{m+1}^2 e_m e_m^T = L_m L_m^T. \tag{3.45}$$

where L_m differs from $\bar{L}_m$ only in the lower right element. They can be related by a scaling matrix $D_m = \text{diag}(1, \ldots, 1, c_m)$, $\bar{L}_m = D_m L_m$. The constraint (3.44) becomes

$$L_m^T y_m = \beta_1 D_m Q_m e_1 = t_m, \tag{3.46}$$

where it is used that L_m is nonsingular and $\beta_1 = \|r_0\|_2$.

There is no need to compute and store matrix Q_k. Instead, the right hand side t_m can be constructed by iteratively applying the Givens rotations to e_1. Finally, the approximate solution x_m is computed by by solving $x_m = V_m L_m^{-T} t_m$ iteratively.

3.2.9 Conjugate gradient method

The conjugate gradient method is one of the best Krylov subspace methods for solving large sparse systems $Ax = b$ where A is symmetric positive definite. It is equivalent to the full orthogonalisation method in the sense that they both share the same Krylov subspace and the subspace of constraints, hence they generate the same approximate solution using exact arithmetic. However, due to the symmetry and definitness of A various simplifications can be made. As a consequence, the required memory per iteration is fixed and the residual and approximate solution are continously updated.

Outline of the method

The conjugate gradient method constructs iteratively a basis $p_0, \ldots, p_m$ for the Krylov subspace such that all vectors are A-orthogonal. The approximate solution after $m+1$ iterations can uniquely be written as a linear combination of these basis vectors,

$$x_{m+1} = x_0 + \sum_{j=0}^{m} \alpha_j p_j = x_m + \alpha_m p_m, \tag{3.47}$$

and the residual is given by the recurrence

$$r_{m+1} = r_m - \alpha_m A p_m. \tag{3.48}$$

The coefficients α_j will be determined by constraining the residual to be orthogonal to the Krylov subspace $\mathcal{K}_{m+1}$. Assume the vectors $r_0, \ldots, r_m$, obtained in previous iterations, are orthogonal and span $\mathcal{K}_{m+1}$. Then r_m is orthogonal to $\mathcal{K}_m$ by assumption and Ap_m is orthogonal to $\mathcal{K}_m$. This leaves one equation to determine coefficient α_m,

$$r_m - \alpha_m A p_m \perp r_m. \tag{3.49}$$

Note that r_{m+1} is obtained using only the last residual, r_m, and the last search direction, p_m. Besides, the residual r_{m+1} is A-orthogonal to $\mathcal{K}_m$ — to see this, compute the inproduct of (3.48) with any $r_j, j < m$.

Theorem 3.4. *TODO*

Proof. Assume $\{p_0, \ldots, p_m\}$ is a set of A-orthogonal vectors with $p_0 = r_0$ and such that the first j vectors span the $j+1$-th Krylov subspace $\mathcal{K}_{j+1}(A, r_0)$, for all $j \leq m$. The last vector of this set, p_m, is of course contained in $\mathcal{K}_{m+1}$, hence can be written as a linear combination of $r_0, \ldots, A^m r_0$. Multiplying p_m to the left by A yields a linear combination of the vectors $Ar_0, \ldots, A^{m+1}r_0 \in \mathcal{K}_{m+2}$. However, $Ap_m \perp \mathcal{K}_m$ by assumption. Hence $\text{span}\{r_m, Ap_m\} \oplus \mathcal{K}_m = \mathcal{K}_{m+2}$. □

Since r_{m+1} is a linear combination of $r_m, Ap_m \in \mathcal{K}_{m+2} \cap \mathcal{K}_m^{\perp}$ and r_{m+1} orthogonal to r_m, it extends the orthogonal basis $\{r_0, \ldots, r_m\}$ to $\mathcal{K}_{m+2}$. Hence, this vector can also be used to generate the next search vector p_{m+1}. As noted earlier, r_{m+1} is already A-orthogonal to $\mathcal{K}_m$. Leaves the A-orthogonalisation with p_m:

$$p_{m+1} = r_{m+1} + \beta_m p_m \perp Ap_m. \tag{3.50}$$

It turns out that the coefficients α_m and β_m can be written as

$$\alpha_m = \frac{\|r_m\|^2}{\|p_m\|_A^2}, \tag{3.51}$$

and

$$\beta_m = \frac{\|r_{m+1}\|^2}{\|r_m\|^2}. \tag{3.52}$$

For a derivation, see Saad (1996, pp. 177–178)

Analysis

The conjugate gradient method is a remarkably simple method for solving a matrix vector problem. The amount of memory and work per iteration is fixed and in theory the exact solution is guaranteed in n iterations, where n is the size of matrix A. Furthermore, it can be shown that CG stops in at most k iterations where k is the number of distinct eigenvalues of A. In practice, rouding errors will spoil the orthogonality of the basis vectors and the optimality is lost. Besides, the number of iterations needed to complete the method is in many cases too large.

The convergence of CG is in theory linear and depends on the condition number of the matrix. However, in many cases the convergence becomes more than linear after a couple of iterations. This is behaviour is called superlinear convergence and is a consequence of eliminating the extreme eigenvalues.

3.3 Preconditioners

The efficiency of all methods discussed above depend heavily on the distribution of eigenvalues of matrix A. In general, the methods exhibit high convergence

rates when the eigenvalues are tightly clustered. For instance, the rate of convergence of the conjugate gradient method is inversely related or bounded by the spectral condition number,

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}. \tag{3.53}$$

To be more precise, the error after m iterations is bounded by (Saad 1996)

$$\|x - x_m\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^m \|x - x_0\|_A, \tag{3.54}$$

where x is the solution of $Ax = b$.

The condition of the system or the distribution of eigenvalues can be improved by multiplicating the system with some invertible matrix M, yielding a new system with the same solution and — by proper construction of M — favourable properties. There are three ways to create such a system: by left preconditioning

$$M^{-1}Ax = M^{-1}b, \tag{3.55}$$

right preconditioning

$$AM^{-1}y = b, \qquad Mx = y, \tag{3.56}$$

or a combination.

The inverse of M should be cheap, or at least cheaper than the inverse of A, since M^{-1} is used every iteration of the iterative method. Furthermore the spectral radius of $I - M^{-1}A$ or $I - AM^{-1}$ should be small. To see this, consider the splitting $A = M - N$ and recall that the rate of convergence of a basic iterative method depends on the spectral radius of $I - M^{-1}A$ (Section 3.1). These properties are the same as for the basic iterative methods Indeed, every basic iterative method originating from a matrix splitting $A = M - N$ can be used as a preconditioner. As such, the basic iterative method is used as an accelerator for a Krylov subspace method, or vice versa.

Probably the most simple preconditioner is diagonal scaling, $M = \mathrm{diag}(A)$. The computational cost and memory requirements are minimal to zero. More advanced preconditioners are based on incomplete LU or Cholesky decompositions of A. The decompositions are for example such that the sparsity structure of the matrix is maintained.

3.4 Deflation

Let A be a symmetric positive definite matrix, possibly preconditioned. Recall that the approximate solution of a Krylov subspace method can be written as

$$\tilde{x} = x_0 + Vy, \tag{3.57}$$

where the columns of V form a basis for the Krylov subspace $\mathcal{K}$ and y is chosen such that the residual

$$r = r_0 - AVy, \tag{3.58}$$

is orthogonal to some subspace $\mathcal{L}$.

Now consider the following extension to the approximate solution $\tilde{x}$,

$$\tilde{x} = x_0 + Vy + Zk, \tag{3.59}$$

with residual

$$r = r_0 - AVy - AZk. \tag{3.60}$$

Here, Z is a matrix whose columns are linearly independent. These columns can be viewed as an augmentation of the Krylov subspace and are called deflation vectors. Applying the same constraint on the residual as used for the derivation of Krylov subspace methods like FOM — the subspace of constraints is equal to the subspace of deflation vectors $\text{col}(Z)$ — yields $r \perp Z$ and

$$Z^T AZk = Z^T(r_0 - AVy). \tag{3.61}$$

Since A is symmetric positive definite $Z^T AZ$ is non-singular and the residual may be written as a projection of the 'original' residual (3.58),

$$r = P(r_0 - AVy), \tag{3.62}$$

where P is the projection defined by

$$P = I - AZ(Z^T AZ)^{-1} Z^T. \tag{3.63}$$

It is easy to verify that P is a projection by calculating the square of P. Using this projection the approximate solution is split in the following way,

$$\tilde{x} = (I - P^T)\tilde{x} + P^T\tilde{x}. \tag{3.64}$$

The first term of the right hand side

$$(I - P^T)x = Z(Z^T AZ)^{-1} Z^T Ax = Z(Z^T AZ)^{-1} Z^T b, \tag{3.65}$$

is an equation independent of the Krylov subspace and can be computed immediately. Using the identity $AP^T = PA$, the second term of (3.64) is found by computing

$$PA\hat{x} = Pb, \tag{3.66}$$

and premultiplying the solution $\hat{x}$ with P^T,

$$P^T\tilde{x} = P^T\hat{x}. \tag{3.67}$$

Chapter 4

Solution techniques for saddle point problems

The Stokes equations belong to the familly of saddle point problems, which are characterised by the following block structure

$$\mathcal{A} = \begin{bmatrix} F & B^T \\ B & \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}, \tag{4.1}$$

where F is an $n \times n$ matrix and B an $m \times n$ matrix. In case of the Stokes equations F is symmetric positive definite and B is of full rank. For reference, the Stokes problem consists of the following subsystems: a velocity subsystem

$$Fu + B^T p = f, \tag{4.2}$$

and a pressure subsystem

$$Bu = 0. \tag{4.3}$$

In Section 4.1 the Schur complement matrix is derived and several approximations are discussed. Section 4.2 presents two direct methods for solving the saddle point system. In Section 4.3 iterative methods and preconditioners are discussed.

4.1 Approximating the Schur complement

Block Cholesky decomposition of $\mathcal{A}$

$$\mathcal{A} = \mathcal{L}\mathcal{D}\mathcal{L}^T = \begin{bmatrix} I & \\ BF^{-1} & I \end{bmatrix} \begin{bmatrix} F & \\ & S \end{bmatrix} \begin{bmatrix} I & F^{-1}B^T \\ & I \end{bmatrix}, \tag{4.4}$$

where S is the Schur complement defined by

$$S = -BF^{-1}B^T. \tag{4.5}$$

The Schur complement S is in general dense, hence it is too expensive to create and store this matrix. Instead, a system with (4.5) is solved using a matrix-free solver, i.e. a solver which only requires the matrix vector product with S, not the matrix S itself. A product with the Schur complement is expensive, since it requires two matrix vector products and one matrix inverse. Furthermore, the Schur complement may be ill-conditioned (Benzi, Golub, and Liesen 2005). Preconditiong a system with S can improve the convergence.

Assume the composition of the diffusion matrix F and the gradient matrix B^T can be approximated by the gradient matrix B^T and some other invertible matrix F_p, to be determined.

$$FB^T \approx B^T F_p \tag{4.6}$$

This approximation will be justified shortly. Premultiplication of (4.6) with BF^{-1} and postmultiplication with F_p^{-1} yields

$$BB^T F_p^{-1} \approx BF^{-1}B^T = -S. \tag{4.7}$$

Note that BB^T is equal to a Laplace matrix and can be easily solved using a Poisson solver. Both matrices BB^T and F_p are square and small compared to the three matrices in (4.5). The inverse of the approximation of S requires one Poisson solve and one matrix multiplication.

4.1.1 Constant viscosity

What is a good approximation of the Schur complement? Consider again the product of the diffusion operator $\mathcal{F}$ and the gradient operator $\mathcal{G}$,

$$(\mathcal{F}\mathcal{G}p)_\alpha = 2(\nu p_{,\alpha\beta})_{,\beta}. \tag{4.8}$$

First, assume the viscosity is constant. Equation (4.8) simplifies to

$$(\mathcal{F}\mathcal{G}p)_\alpha = 2\nu p_{,\alpha\beta\beta} = (2\nu p_{,\beta\beta})_{,\alpha} = \mathcal{G}(2\nu\mathcal{G}^*\mathcal{G})p. \tag{4.9}$$

This implies an operator $\mathcal{F}_p = 2\nu\mathcal{G}^*\mathcal{G}$, or in discrete terms $F_p = 2\nu BB^T$. This choice of F_p shows that the Schur complement is determined by a scaled identity matrix,

$$S = -2BB^T(BB^T)^{-1}\nu^{-1} = -2\nu^{-1}I. \tag{4.10}$$

4.1.2 Pressure convection diffusion

$$(\mathcal{F}\mathcal{G}p)_\alpha = 2(\nu p_{,\alpha\beta})_{,\beta} = 2(\nu p_{,\beta})_{,\beta\alpha} - 2(\nu_{,\alpha}p_{,\beta})_{,\beta} = \mathcal{G}(2\mathcal{G}^*\nu\mathcal{G}p) - 2(\nu_{,\alpha}p_{,\beta})_{,\beta}. \tag{4.11}$$

Neglecting the second term yields an operator $\mathcal{F}_p = 2\mathcal{G}^*\nu\mathcal{G}$, which is very similar to $\mathcal{F}$. The neglected term indicates that the approximation is at least accurate for slowly varying viscosity. When used as a preconditioner, this approach is called the *pressure convection diffusion* method.

4.1.3 Least-squares commutator

The above choice of F_p is made intuitively. The *least-squares commutator preconditioner* defines F_p in a more formal way by minimising the commutator, i.e. FB^T and $B^T F_p$,

$$F_p = \arg\min_{\hat{F}_p} \left\| FB^T - B^T \hat{F}_p \right\|_2 . \tag{4.12}$$

This yields $F_p = -(BB^T)^{-1}BFB^T$ and the Schur complement is approximated by

$$S \approx -(BB^T)(BFB^T)^{-1}(BB^T). \tag{4.13}$$

Convergence is improved by applying scaling matrices M_1 and M_2 in the following way,

$$S \approx -(BM_2^{-2}B^T)(BM_2^{-2}FM_1^{-1}B^T)^{-1}(BM_1^{-1}B^T). \tag{4.14}$$

In Rehman (2010) the scaling matrices are chosen to be $M_1 = M_2^2 = \mathrm{diag}(F)$. In terms of work this approximation is more expensive than the pressure convection diffusion method.

4.2 Direct methods

4.2.1 Schur complement reduction

Premultiplying (4.1) with the inverse of the block lower triangular matrix $\mathcal{L}$ gives

$$\begin{bmatrix} F & B^T \\ & S \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ -BF^{-1}f \end{bmatrix}. \tag{4.15}$$

Now the pressure p can be solved independently of the velocity u. Applying back substitution yields the pressure subsystem,

$$-Sp = BF^{-1}B^T p = BF^{-1}f, \tag{4.16}$$

and the velocity subsystem,

$$Fu = f - B^T p. \tag{4.17}$$

As noted earlier, a system with the Schur complement S is solved using a matrix-free solver. Alternatively, (4.16) can be solved using a generalised least-squares method (Benbow 1999). Equation (4.16) is equivalent to

$$\text{find} \quad p \in \mathbb{R}^m \quad \text{such that} \quad q^T F^{-1}(B^T p - f) = 0 \quad \forall q \in \mathrm{row}(B), \tag{4.18}$$

where $\mathrm{row}(B)$ is the subspace of columns of B. By assumption F is symmetric positive definite, hence F^{-1} is SPD as well and defines an inner product $\langle \cdot, \cdot \rangle_{F^{-1}}$ by $\langle v, w \rangle_{F^{-1}} = v^T F^{-1} w$. Substituting the inner product in (4.18) yields

$$\text{find} \quad p \in \mathbb{R}^m \quad \text{such that} \quad \left\langle q, B^T p - f \right\rangle_{F^{-1}} = 0 \quad \forall q \in \mathrm{row}(B). \tag{4.19}$$

Since $p^T B \in \text{row}(B)$, $B^T p$ is the F^{-1}-orthogonal projection of f onto $\text{row}(B)$. This implies that p is the result of the minimisation of $B^T p - f$ in F^{-1}-norm,

$$p = \underset{p_* \in \mathbb{R}^m}{\arg\min} \|B^T p_* - f\|_{F^{-1}}. \tag{4.20}$$

This system can be solved using a generalisation of the least-squares method, LSQR(F^{-1}) (Benbow 1999).

Numerical experiments in Benbow (1999) show that solving (4.20) with LSQR(F^{-1}) needs less iterations to acchieve a certain accuracy than MINRES applied to (4.16), if the Schur complement S is ill conditioned. If the inverse of F is relatively cheap, than the LSQR(F^{-1}) approach is faster in time than MINRES.

4.2.2 Null space method

The pressure subsystem (4.3) constrains the velocity u to the kernel or null space of B, $u \in \ker(B)$. By assumption B is a full rank $m \times n$-matrix, hence the kernel is of dimension $n-m$. Let $\{z_1, \ldots, z_{n-m}\}$ be a basis for the $\ker(B)$, then the velocity u can be written as a linear combination of these basis vectors,

$$u = Zv, \tag{4.21}$$

where $Z = [z_1, \ldots, z_{n-m}]$ and $v \in \mathbb{R}^{n-m}$. This representation automatically satisfies equation (4.3).

Substituting (4.21) for u in (4.2) yields

$$FZv + B^T y = f. \tag{4.22}$$

Note that the rows of B are orthogonal to the kernel of B. The orthogonal projection of (4.22) onto $\ker(B)$ reduces (4.22) to a system without pressure y,

$$\text{proj}_{\ker(B)} \left(FZv + B^T p\right) = \text{proj}_{\ker(B)} \left(FZv\right) = \text{proj}_{\ker(B)} f. \tag{4.23}$$

Since the columns of Z span the kernel of B, this is equivalent to

$$Z^T F Z v = Z^T f. \tag{4.24}$$

By assumption, matrix F is symmetric positive definite, hence $Z^T F Z$ is SPD as well. The pressure y is found by solving (4.22) using the solution v of (4.24).

By assumption, matrix B represents a discrete divergence operator on a rectangular or cubic domain with periodic boundary conditions — ignoring the shifting of boundaries here. Hence, an orthogonal basis $z_1, \ldots, z_{n-m}$ is easily constructed by considering fourier modes. Let ϕ^j be a fourier mode with frequency k^j_α in direction α,

$$\phi^j = \exp(2\pi i x_\alpha k^j_\alpha). \tag{4.25}$$

Applying the divergence operator to ϕ^j yields

$$\phi^j_{\alpha,\alpha} = 2\pi i k^j_\alpha \mathbf{1}_\alpha \phi^j = 0 \quad \iff \quad k^j_\alpha \mathbf{1}_\alpha = 0. \tag{4.26}$$

where $\mathbf{1}_\alpha = 1$.

A special choice of basis vectors leads to a very simple system to be 'solved' for the velocity u. Let $z_1, \ldots, z_{n-m}$ be an F-orthogonal basis of $\ker(B)$. Then $Z^T F Z$ is a diagonal matrix and the velocity u is simply given by

$$u = (Z^T F Z)^{-1} Z Z^T f = \sum_{j=1}^{n-m} \frac{\langle f, z_j \rangle}{\langle z_j, z_j \rangle_F} z_j. \tag{4.27}$$

This is easily transformed into an iterative scheme where the basis vectors are constructed one by one until a desired accuracy is reached,

$$u_{j+1} = u_j + \frac{\langle f, z_{j+1} \rangle}{\langle z_{j+1}, z_{j+1} \rangle_F} z_{j+1}. \tag{4.28}$$

4.3 Iterative methods and preconditioners

Consider the splitting of $\mathcal{A}$,

$$\mathcal{A} = \mathcal{M} - \mathcal{N}. \tag{4.29}$$

This splitting leads to the following stationary iteration (Wesseling 2010, Section 7.6)

$$\begin{bmatrix} u_{k+1} \\ p_{k+1} \end{bmatrix} = \begin{bmatrix} u_k \\ p_k \end{bmatrix} + \mathcal{M}^{-1} \left(\begin{bmatrix} f \\ 0 \end{bmatrix} - \mathcal{A} \begin{bmatrix} u_k \\ p_k \end{bmatrix} \right). \tag{4.30}$$

A good splitting results in a matrix $\mathcal{M}$ which is cheap to invert (implicitly) and, at the same time, close to $\mathcal{A}$ in some sense.

The same splitting can be used to precondition the original system and solve using a Krylov subspace method. Preconditioner

$$\mathcal{M}^{-1} \mathcal{A} \begin{bmatrix} u \\ p \end{bmatrix} = \mathcal{M}^{-1} \begin{bmatrix} f \\ 0 \end{bmatrix}$$

or postconditioner/right-preconditioner

$$\mathcal{A} \mathcal{M}^{-1} \begin{bmatrix} u' \\ p' \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}, \quad \mathcal{M}^{-1} \begin{bmatrix} u' \\ p' \end{bmatrix} = \begin{bmatrix} u \\ p \end{bmatrix}$$

4.3.1 Schur method

Using the block Cholesky decomposition (4.4) the saddle point problem (4.1) can be written as

$$(\mathcal{L}\mathcal{D}) \mathcal{L}^T \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}. \tag{4.31}$$

Applying block forward subsitution using $\mathcal{L}\mathcal{D}$ and block backward subtitution using $\mathcal{L}^T$ solves the problem. This involves solving subsystems with matrices F and S. The accuracy of the final solution depends on the accuracy used in solving the subsystems. In general, solving subsystems with the same accuracy

as desired for the final solution is not enough (Rehman 2010, Chapter 8). Choosing the right accuracy to reach a decent solution while keeping work as low as possible may be difficult.

To overcome this problem, an iterative method is suggested in Rehman (2010, Chapter 8), where all subsystems are solved using an arbitrary accuracy, say ϵ, possibly lower than required for the final solution. This can be written as the splitting with matrix $\mathcal{M}$ given by

$$\mathcal{M}_{\text{Schur}} = \begin{bmatrix} \tilde{F} & \\ B & \tilde{S} \end{bmatrix} \begin{bmatrix} I & \tilde{F}^{-1}B^T \\ & I \end{bmatrix} \tag{4.32}$$

where the tildes mean that corresponding subsystems should be solved with accuracy ϵ. One iteration of (4.30) with $\mathcal{M}_{\text{Schur}}$ requires the solution of three subsystems,

$$\tilde{F}u_* = f - Fu_k - B^T p_k, \tag{4.33}$$

$$\tilde{S}(p_{k+1} - p_k) = -B(u_k + u_*), \tag{4.34}$$

and

$$u_{k+1} = u_k + u_* - \tilde{F}^{-1}B^T p_*. \tag{4.35}$$

The most expensive part of the Schur method is (4.34). Solving a system with the Schur complement S takes several matrix vector products with S. Every product with S requires the solution of a subsystem with F.

In Rehman (2010, Chapter 8) the Schur method is tested on the Stokes equations, where all subsystems are solved with the same accuracy as required for the final solution. For the isoviscous case, one iteration is sufficient to reach the final accuracy. When the viscosity is variable, then depending on the viscosity contrast an extra iteration is required. The number of iterations needed to solve the Schur complement system (4.34) is almost independent of grid size and Reynolds number.

4.3.2 SIMPLE

In the Schur method, every iteration a system with the Schur complement S is solved. As noted earlier, the Schur complement is not constructed, but solved implicitly using a matrix-free solver, which only uses the matrix vector product with S. This product, however, requires the inverse of F. The Semi-Implicit Method for Pressure-Linked Equations, SIMPLE for short, approximates the inverse of F by the inverse of the diagonal of F.

Applying the approximation of F^{-1} to $\mathcal{A} = (\mathcal{L}\mathcal{D})\mathcal{L}^T$ yields a splitting with matrix $\mathcal{M}$ given by

$$\mathcal{M}_{\text{SIMPLE}} = \mathcal{L}_{\text{SIMPLE}}\mathcal{U}_{\text{SIMPLE}} = \begin{bmatrix} F & \\ B & -B\tilde{F}^{-1}B^T \end{bmatrix} \begin{bmatrix} I & \tilde{F}^{-1}B^T \\ & I \end{bmatrix}, \tag{4.36}$$

where $\tilde{F} = \text{diag}(F)$ the diagonal of F. As in the Schur method, a system with $\mathcal{M}_{\text{SIMPLE}}$ is to be solved by applying block forward subsitution using the first matrix, $\mathcal{L}_{\text{SIMPLE}}$ and backward substitution using the second matrix, $\mathcal{U}_{\text{SIMPLE}}$.

This yields equations similar to (4.33)–(4.35), however, the simplifications result in much less work.

The convergence of SIMPLE as an iterative method is slow. Faster convergence can be achieved when SIMPLE is used as a preconditioner for a Krylov subspace method. However, the rate of convergence depends on the grid size and Reynolds number (Rehman 2010, Chapter 6).

4.3.3 SIMPLER

When SIMPLE is used as a preconditioner, the dependence on the Reynolds number is removed by extending every iteration of SIMPLE in the following way. Given the solution after k steps, u_k and p_k, an intermediate solution u_*, p_* is generated by solving a system similar to (4.36),

$$\mathcal{M}_{\text{SIMPLE}}^T \begin{bmatrix} u_* - u_k \\ p_* - p_k \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix} - \mathcal{A} \begin{bmatrix} u_k \\ p_k \end{bmatrix}, \tag{4.37}$$

which is syntactic sugar for 'apply forward substitution using $\mathcal{U}_{\text{SIMPLE}}^T$ and backward substitution using $\mathcal{L}_{\text{SIMPLE}}^T$.' Then a SIMPLE step is appied to u_* and p_*,

$$\mathcal{M}_{\text{SIMPLE}} \begin{bmatrix} u_{k+1} - u_* \\ p_{k+1} - p_* \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix} - \mathcal{A} \begin{bmatrix} u_* \\ p_* \end{bmatrix}, \tag{4.38}$$

resulting in u_{k+1} and p_{k+1}.

This approach, called SIMPLER, doubles the amount of work per iteration compared to SIMPLE. On the other hand, the iterative method will converge faster. When used as a preconditioner, the first subsystem of (4.37) doesn't significantly improve convergence. The original SIMPLER method discards the first subsystem by setting $u_* = u_k$. This leaves only the second subsystem of (4.37),

$$B\tilde{F}^{-1}B^T y_* = B\tilde{F}^{-1}((\tilde{F} - F)x_k + f). \tag{4.39}$$

The amount of work in solving one iteration of SIMPLER, (4.39) and (4.38), is roughly 1.3 times the amount of work in solving one iteration of SIMPLE, (4.36).

4.3.4 Block triangular preconditioner

Block triangular preconditioners have the following form,

$$\mathcal{P}_t = \begin{bmatrix} \tilde{F} & B^T \\ & \tilde{S} \end{bmatrix}, \tag{4.40}$$

where $\tilde{F}$ and $\tilde{S}$ are approximations to F and S respectively. A simple choice for $\tilde{F}$ is the diagonal of F. The Schur complement matrix may be approximated by an identity matrix, $\tilde{S} = I$.

The *pressure convection diffusion* preconditioner uses the exact matrix F and the approximation for the Schur complement given in Section 4.1.2. Numerical

experiments in Rehman (2010, Chapter 4) on a Navier-Stokes problem show that the rate of convergence of a Krylov subspace method in combination with the pressure convection diffusion preconditioner is independent of the mesh size.

The *least-squares commutator* preconditioner uses the approximation for the Schur complement given in Section 4.1.3. Numerical experiments in Rehman (2010, Chapter 4) show that the rate of convergence is mildly dependent on the mesh size. However, this method converges faster than the pressure convection diffusion preconditioner.

Chapter 5

Current implementation

This chapter presents the current implementation for solving the Stokes equations (Section 5.1). In Section 5.2 two test problems are defined. Using one of these problems the current implemtation is evaluated (Section 5.3). The results are compared with a MINRES solver applied to the same test.

5.1 Algorithms

The steepest descent method, described in Section 3.1.2, is an iterative method for solving linear systems. The same method is also suitable for the minimisation of non-linear functions. Let $\phi : \mathbb{R}^n \to \mathbb{R}$ be some non-linear function and x_0 an arbitrary initual guess. The next iterate is searched, starting from x_0, in the direction of steepest descent, the gradient of ϕ, such that ϕ is minimised. For linear systems the minimisation is simply determined by two inner products. However, for non-linear functions the minimisation is done by literally searching along the line of steepest descent for a point below a certain threshold, using a so called line search method. The threshold is the value of ϕ of the previous iterate, $\phi(x_{k-1})$, hence the line search method should return a new approximate solution closer to the minimum of ϕ. The algorithm of the steepest descent method with line search is given in Algorithm 5.1. Another method implemented in Culgi is the closely related Polak-Ribi‘ere method (Algorithm 5.2).

A simple line search method works as follows. The steepest descent method feeds the line search with a previous solution x_k and a search direction d_k. The first candidate solution is located at $x_k + \alpha d_k$, where α is initially a fixed number or obtained from a previous line search. The function ϕ is evaluated for this candidate solution. If the $\phi(x_k + \alpha d_k) < \phi(x)$, then the candidate solution is accepted, $x_{k+1} = x_k + \alpha d_k$. Otherwise α is divided by zero and the process is repeated. This procedure is illustrated in Figure 5.1. The algorithm is given in Algorithm 5.3. Another closely related line search method, implemented in Culgi, is given in algorithm 5.4. Culgi two line search methods are implemented.

Culgi lacks support for matrices. The matrix vector product is computed using various discrete operators, mimicing the analytical operators of the Stokes

Algorithm 5.1: Steepest descent with line search

```
1  steepest_descent(x_0; n, ε, δ):
2      r_0 := Ax_0 − b
3      for k = 0, 1, 2, ..., n:
4          x_{k+1} := line_search(x_k, r_k)
5          r_{k+1} := b − Ax_k
6          if ‖r_{k+1}‖_∞ < ε or ‖r_k‖_∞ − ‖r_{k+1}‖_∞ < δ:
7              return x_{k+1}
```

Algorithm 5.2: Polak-Ribière with line search

```
1  Polak-Ribière(x_0; n, ε, δ):
2      r_0 := Ax_0 − b
3      d_0 := r_0
4      for k = 0, 1, 2, ..., n:
5          x_{k+1} := line_search(x_k, d_k)
6          r_{k+1} := b − Ax_k
7          if ‖r_{k+1}‖_∞ < ε or ‖r_k‖_∞ − ‖r_{k+1}‖_∞ < δ:
8              return x_{k+1}
9          β_k := max(0, ⟨r_{k+1}, r_{k+1} − r_k⟩ / ⟨r_k, r_k⟩)
10         d_{k+1} := r_{k+1} + β_k d_k
```

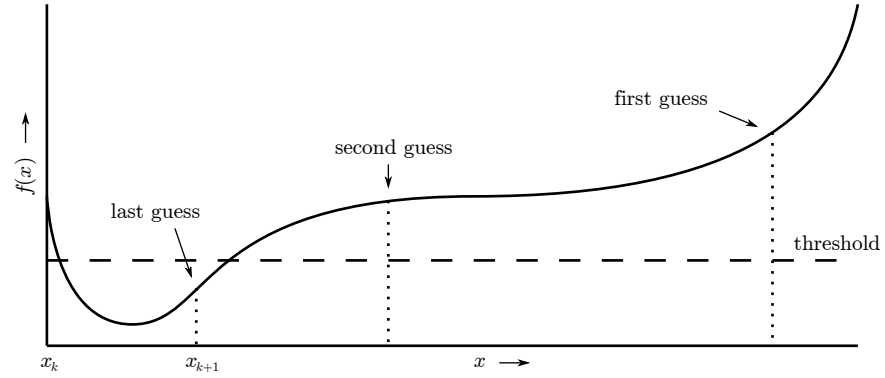

Figure 5.1: Illustration of a line search method. The function f, to be minimised, is evaluated at some initial point, the first guess. Since this point is above the threshold, this guess is rejected. The second guess is in the middle of x_k and the previous guess. The third guess is below the threshold and is accepted as the new approximate solution, x_{k+1}.

Algorithm 5.3: Simple line search, illustrated by Figure 5.1

```
1  line_search(x_k, d_k; m, α):
2      r_k := b − A x_k
3      for j = 0, −1, −2, ..., −m + 1:
4          if ‖r_k − α 2^j A d_k‖_∞ < ‖r_k‖_∞:
5              return x_k + α 2^j d_k
6      return x_k + α 2^{−m} d_k
```

equations. The boundary conditions are imposed by adding grid points outside the domain to all vectors, velocity, pressure, viscosity, et cetera, and applying the periodicity to these enlarged vectors using the definitions of Section 2.5.

5.2 Test problems

5.2.1 Simulation of an oil droplet in water

The first test problem is a mixture of oil and water. In initial state, the oil is suspended as a droplet (sphere) in the center of the domain (see Figure 5.2). Oil and water repel each other, hence the initial state is close to equilibrium. Furthermore, both oil and water have different viscosity coefficients. The sharpness of the interface between oil and water is controlled by adjusting the diffusion rate of oil and water and the amount of repulsion. These coefficients are of course determined by nature, however, for the sake of testing any choice of coefficients is allowed.

Algorithm 5.4: Armijo line search

```
line_search(x_k, d_k; m, α):
    r_k := b − A x_k
    for j = 0, −1, −2, ..., −m + 1:
        if ‖r_k − α 2^j A d_k‖_2^2 < (1 − α 2^j 10^{−4}) ‖r_k‖_2^2:
            return x_k + α 2^j d_k
    return x_k + α 2^{−m} d_k
```

.

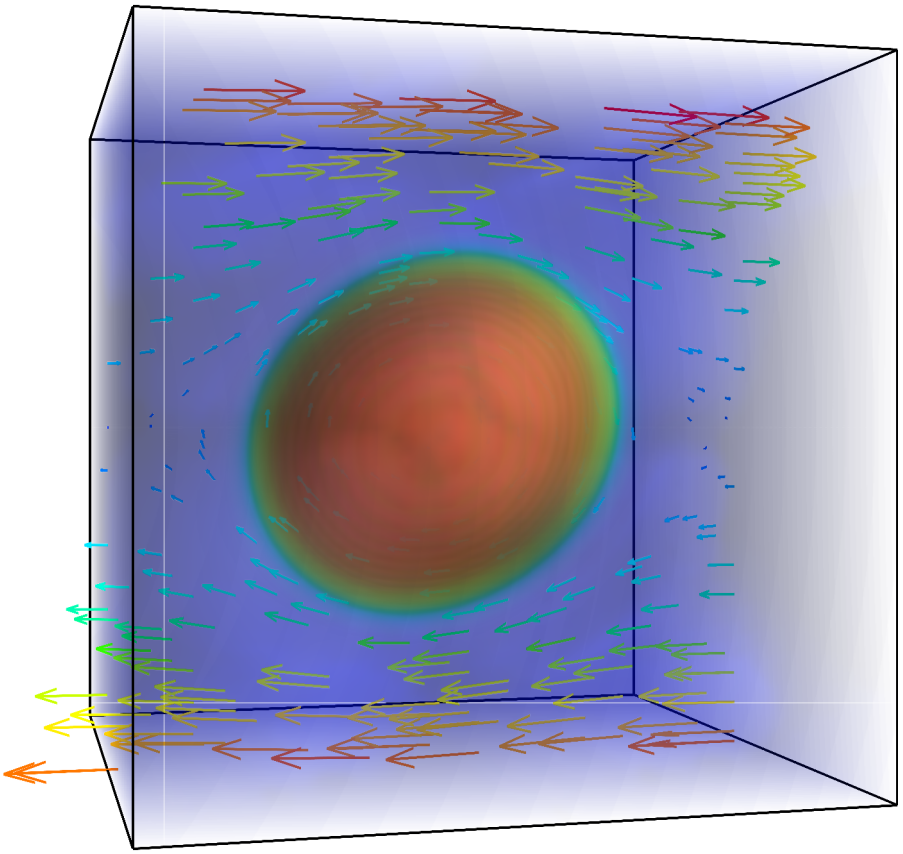

Figure 5.2: Volume rendering of an oil droplet suspended in water under influence of a shear velocity after several time steps. Red means a high oil density and transparant/blue a low density. Since the sum of densities of water and oil is constant, blue implies a high density of water. The velocity is displayed on a plane at the center of the domain. Inside the oil droplet the velocity is circulating, which shows that the oil droplet rotates in time.

5.2.2 Diblock copolymer blend

The second test problem is a mixture of polymers, consisting of molecules of some type $\mathcal{A}$ and $\mathcal{B}$. The polymers are created in a reaction process where all molecules of any type may react with each other, forming a (small) polymer $\mathcal{A}A$, $\mathcal{A}B$ or $\mathcal{B}B$. These newly formed polymers are able to react with other molecules and polymers, creating longer and longer chains.

Since molecules of type $\mathcal{A}$ and $\mathcal{B}$ repel each other, $\mathcal{A}$-molecules are more likely to react with other $\mathcal{A}$-molecules and the same is true for $\mathcal{B}$. After a while, a polymer may look like $\mathcal{A}AABAABABBBBABBB$. There is a clear distinction between an $\mathcal{A}$-part and a $\mathcal{B}$-part, which is typical for all polymers in this reaction. This phenomenon leads to the approximation of all these polymers by $\mathcal{A}AAAAAAABBBBBBBB$, conveniently written as $\mathcal{A}8B8$.

The end of the above described reaction process is the beginning of the test problem. The initial phase is a chaotic configuration of the polymers. The $\mathcal{A}$-tails of the polymers tend to cluster due to the repulsion of $\mathcal{B}$. This choactic clustering is clearly visible in Figure 5.3. Under influence of a shear velocity other configurations become favourable. Laminar phases may occur, where the polymers are nicely ordered in planes, or cylindrical phases, where $\mathcal{A}$-tails cluster in cylinders while the $\mathcal{B}$-tails fill the surrounding space.

5.3 Numerical results

The first test problem is used to investigate the Polak-Ribière method with Armijo line search, together refered to as NLE solver, non-linear equation solver. The number of outer iterations (Polak-Ribiére) is limited by 10000, the number of line search iterations by 5 and the previous α is used (multiplied by two) as a first guess for the line search method. The same tests are performed using a MINRES solver. The evolution equation is not used in these tests. All tests are performed with the dimensionless viscosity for water equal to 1 and stopping criterion $\|r_k\|_\infty < 10^{-3}$. A two dimensional box is simulated as a three dimensional box with one grid cell in the third dimension. The initial guess for the velocity is equal to a linear shear velocity, the pressure is simply chosen zero.

Table 5.1 lists the number of iterations needed by the solvers to find a solution for different grid sizes, when the dimensionless oil viscosity is 10. The MINRES solver is outperforms the NLE solver for every grid size. For both methods the number of iterations depends on the grid size.

Table 5.2 lists the number of iterations needed by the solvers to find a solution for different oil viscosities. The grid size is 32 in each direction. Again, the MINRES solver is superior to the NLE solver. For large viscosity contrasts, ratio between oil and water viscosity, the NLE solver fails to converge.

Figure 5.4 shows the residual norm versus the number of iterations for a three dimensional box with 64 grid points in every direction and an oil viscosity of 10. This figure shows that increasing the accuracy for the NLE solver will have a great impact on the number of iterations.

.

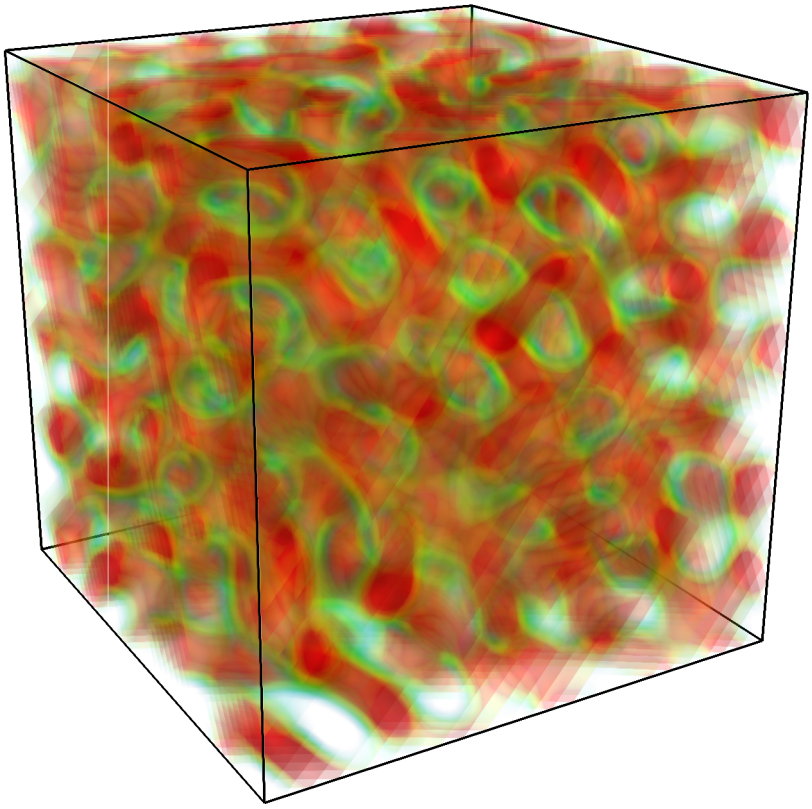

Figure 5.3: A typical snapshot of the initial stages of microphase separation in a symmetric $\mathcal{A}8B8$ polymer blend. Red means a high density of $\mathcal{A}$ molecules, while the transparant/greenish parts indicate a high density of $\mathcal{B}$ molecules.

Table 5.1: Number of iterations used by the solvers for the simulation of an oil droplet suspended in water, where oil has dimensionless viscosity 10.

box type	box size	NLE iter.	NLE l.s. tot. iter.	MINRES iter.
2d	16	346	695	63
	32	1176	2354	172
	64	2798	5598	359
	128	3446	6894	504
3d	16	363	729	99
	32	1083	2169	269
	64	2293	4589	558
	128	4411	8825	770

Table 5.2: Number of iterations used by the solvers for the simulation of an oil droplet suspended in water, where the box has 32 grid points in each direction.

box type	oil viscosity	NLE iter.	NLE l.s. tot. iter.	MINRES iter.
2d	10.0	1176	2354	172
	20.0	2524	5051	248
	40.0	4986	9976	343
	100.0	*did not converge*		498
	1000.0	*did not converge*		1225
3d	10.0	1083	2169	269
	20.0	2115	4234	472
	40.0	4505	9015	676
	100.0	*did not converge*		1057
	1000.0	*did not converge*		2974

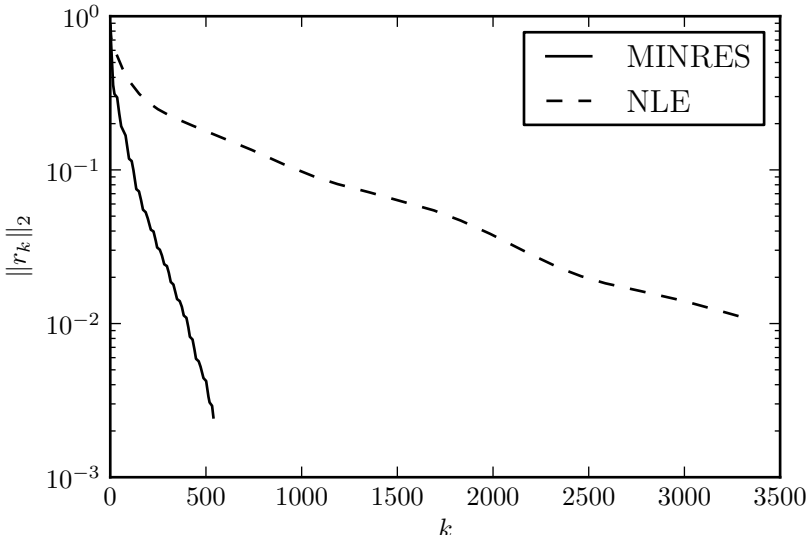

Figure 5.4: Residual norm versus number of iterations k for the simulation of an oil droplet suspended in water on a three dimensional grid with 64 grid points in every direction

When the Stokes equations are used as a part of the evolution equation, the solvers converge much faster for all iterations after the first one, due to the use of the previous solution as an initial guess (Table 5.3). Note that the number of iterations for the MINRES solver is limited to 5000. The last line of the table contains two evolution steps where the MINRES solver reached this limit.

Table 5.3: Number of iterations used by the MINRES solver for the computation of the velocity field in the first five evolution (evo.) steps.

box type	oil viscosity	number of iterations				
		evo. 1	evo. 2	evo. 3	evo. 4	evo. 5
2d	10.0	359	88	32	25	1
	100.0	1484	80	1	62	1
	1000.0	3900	12	8	9	10
3d	10.0	558	27	25	1	1
	100.0	3043	3	5	5	75
	1000.0	5000	5000	671	1	1

Chapter 6

Future research

As mentioned in the previous chapter, the Culgi software package lacks support for operations with sparse matrices. This limits the possible solution techniques, such as incomplete factorisations and the promising Schur method. The first method chosen for future research is the block triangular preconditioner supplemented with a deflation method. The second method which will be investigated is the null space method. The implementation of these methods in the Culgi software package should be feasible.

The block triangular preconditioner uses approximations for the Laplace-like matrix F and the Schur complement S. The most simple approximations are the diagonal of F and an identity matrix for the Schur complement. However, there are many variants, some of which are feasible to implement in Culgi. The choice of these approximations is subject of further investigation.

The deflation method is only briefly discussed in this report. Further research is necessary for the details regarding the implementation. The choice of deflation vectors is very important for the efficiency of the method. As noted in the previous chapter, large viscosity contrasts have a severe implication on the rate of convergence. Deflation vectors based on these viscosity contrasts may enhance the convergence.

The null space method depends on the availability of a basis for the null space of the divergence operator B. For periodic boundaries without shifting due to a shear velocity (see Section 2.5) an analytical basis for the null space can be formulated based on Fourier modes. Wether such an analytical basis can be constructed for shifting boundaries is a topic for further research. Furthermore, it is yet unclear if the null space can compete with preconditioned Krylov subspace methods if such a basis exists.

As noted in the previous chapter, the very first computation of the Stokes equations, in absence of a previous solution, requires many iterations compared to the subsequent solves. It might be useful to develop a good initial solution to speed up the first computation. Furthermore, if the previous solution is used as initial guess for the next time step, then it might be useful to apply a convection step using the previously obtained velocity, if such an update is cheap and accurate.

From the numerical results in the previous chapter follows that the grid size and viscosity contrast influence the rate of convergence of MINRES. Furthermore, it is expected that the sharpness of the interface between molecules also plays a role in the rate of convergence. The research into the methods defined above should reflect on these convergence issues. The test problems defined in the previous chapter are suitable to investigate all these issues.

Bibliography

Benbow, Steven J. (1999). "Solving Generalized Least-Squares Problems with LSQR". In: *SIAM Journal on Matrix Analysis and Applications* 21.1, pp. 166–177. DOI: 10.1137/S0895479897321830. URL: http://link.aip.org/link/?SML/21/166/1.

Benzi, Michele, Gene H. Golub, and Jörg Liesen (2005). "Numerical solution of saddle point problems". In: *Acta Numerica* 14.-1, pp. 1–137. DOI: 10.1017/S0962492904000212. eprint: http://journals.cambridge.org/article_S0962492904000212. URL: http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=298722&fulltextType=RA&fileId=S0962492904000212.

Eisenstat, Stanley C., Howard C. Elman, and Martin H. Schultz (1983). "Variational Iterative Methods for Nonsymmetric Systems of Linear Equations". In: *SIAM Journal on Numerical Analysis* 20.2, pp. 345–357. DOI: 10.1137/0720023. URL: http://link.aip.org/link/?SNA/20/345/1.

Golub, G.H. and C.F. Van Loan (1996). *Matrix Computations*. 3rd ed. The Johns Hopkins University Press.

Paige, C.C. and M.A. Saunders (1975). "Solution of Sparse Indefinite Systems of Linear Equations". In: *SIAM Journal on Numerical Analysis* 12.4, pp. 617–629. DOI: 10.1137/0712047. URL: http://link.aip.org/link/?SNA/12/617/1.

Rehman, Mehfooz ur (2010). "Fast Iterative Methods for the Incompressible Navier-Stokes Equations". PhD thesis. Delft University of Technology.

Saad, Y. (1996). *Iterative methods for sparse linear systems*. 1st ed. PWS.

Vorst, H.A. van der and C. Vuik (1994). "GMRESR: a family of nested GMRES methods". In: *Num. Lin. Alg. Appl.* 1, pp. 369–386.

Wesseling, Pieter (2010). *Principles of Computational Fluid Dynamics*. 2nd ed. Springer Series in Computational Mathematics 29. Springer Berlin Heidelberg. ISBN: 978-3-642-05162-3. DOI: 10.1007/978-3-642-05146-3.