Increasing the parallel efficiency of multi-scale, space-time simulations of turbulent flows

intermediate report

Gertjan van Zwieten

December 21, 2005

Preface

This report ends the three-month period of preliminary research for the Master of Science's degree in Applied Mathematics at the faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology. The research was carried out at Habanera, a company that develops numerical software that can be used in computer simulations and other types of scientific computations.

This master project is closely connected to the research project *Variational Multiscale Large-Eddy Simulation for Deforming Domains* at the faculty of Aerospace Engineering of Delft University of Technology. That project uses Habanera's software toolkits Jem and Jive for its simulations of turbulent flows. Existing code will be available during the next six months of this master project, in which the mathematical techniques described at the end of this report will be applied to this problem.

Gertjan van Zwieten

Contents

Chapter 1

Introduction

Predicting fluid-structure interactions is an important aspect in the design of aircraft and civil structures. The cheapest and also most flexible source of data is a numerical simulation of the flow around the structure. Unfortunately, in reality these flows are almost always turbulent, which means that they consist of fluid motions that vary enormously in scale. This puts such high requirements on computing power that a direct numerical simulation of these flows is not feasible, and is not expected to be in the near future. Therefore, several alternative methods are developed that manage to reduce these computing requirements, albeit at the cost of accuracy. This report focuses on a method that was proposed quite recently: the Variational Multiscale (VMS) method.

The purpose of this report is twofold. First, it aims to give a thorough overview of the VMS method, starting from the governing equations for non-steady, compressible flow and leading all the way to an actual implementation of the method. This implementation has already been created and tested by the research group at the faculty of Aerospace Engineering. The tests revealed that under certain conditions, the current implementation of the method converges very poorly. Therefore, the second aim of this report is to introduce a new mathematical technique, deflation, that is expected to improve convergence in these situations.

The report is divided in four parts. Chapter 2 presents a complete derivation of the VMS method. Chapter 3 is a general overview of some important numerical topics, such as Krylov subspace methods, preconditioners and domain decomposition. In Chapter 4 the previous two chapters are combined into a complete implementation of the VMS method. The problems that were experienced are mentioned as well. Section 4.3 presents a detailed introduction to the deflation technique, which is expected to solve some of these problems. The implementation of this technique is future research, subject of Chapter 5.

Chapter 2

Turbulence modeling

In fluid dynamics, a main distinction is that of laminar and turbulent flows. Laminar flows are characterized by a smooth and predictable nature, whereas turbulent flows are more chaotic, with rapid and seemingly random variations in pressure and velocity. The distinctive difference between the two types of flow can be seen in Figure 2.1. In practice, most flows are turbulent. Its chaotic nature makes that this type of flow very hard to simulate numerically, due mostly to the enormous range of scales of motion in such flows.

This chapter presents the Variational Multiscale (VMS) method, which aims to accurately describe turbulent flows at a much lower cost than a direct numerical simulation. Section 2.1 starts with an overview of the governing Navier-Stokes equations. Section 2.2 is an overview of Large Eddy Simulation, which is similar to the newer VMS method that is introduced next in Section 2.3. Section 2.4 defines the numerical representation of the flow, and Section 2.5 shows how this flow is constructed with a discontinuous Galerkin method. Finally, Section 2.6 shows how the resulting non-linear system of equations can be solved.

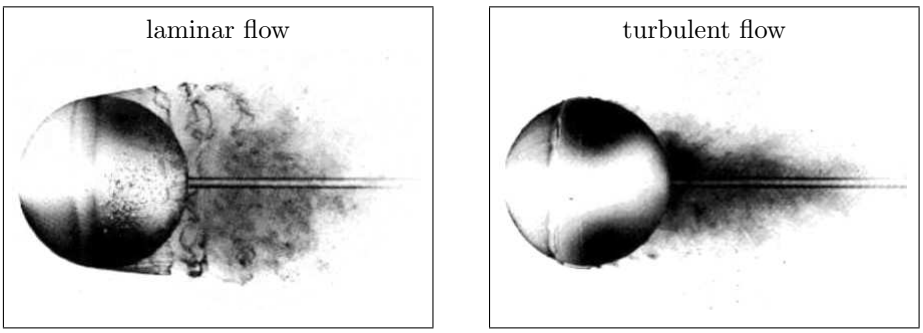

Figure 2.1: Comparison of laminar and turbulent flow past a sphere in water, copied from Anderson [1], Figures 6.9 and 6.10. The laminar flow separates readily from the surface, leaving a large wake. The turbulent flow separates much further back on the surface.

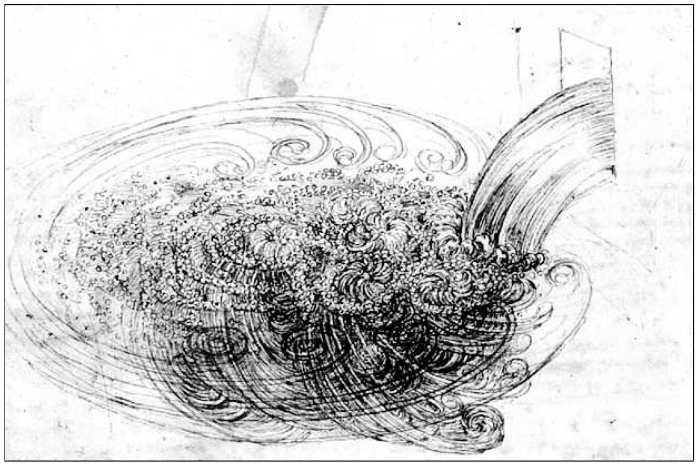

Figure 2.2: Part of Leonardo da Vinci's *Studies of water passing obstacles and falling*, showing a flow field induced by a falling stream. The drawing shows many different scales of fluid motion.

A note on notation: following common practice in fluid dynamics, the equations in this chapter are presented in *Cartesian tensor notation*. This means that spatial and temporal differentiation are denoted as follows:

$$\phi_{,i} = \frac{\partial \phi}{\partial x_i}, \quad \phi_{,t} = \frac{\partial \phi}{\partial t}. \tag{2.1}$$

Also, Greek indices occurring twice within a term or product are summed over that index. This is known as the *Einstein summation convention*. For example, the divergence of a vector field $\boldsymbol{u}(\boldsymbol{x})$ in 3D space is notated as:

$$u_{\alpha,\alpha} = u_{1,1} + u_{2,2} + u_{3,3}. \tag{2.2}$$

2.1 The Navier-Stokes equations

Turbulent flows contain an enormous range of fluid motions. See for example Figure 2.2, the famous Leonardo da Vinci drawing which shows wide circular motions around a sluice and much smaller whorls near the impact of the stream. Even the smallest whorls, however, are about five orders of magnitude larger than the discrete molecules that form the fluid. This molecular nature of the fluid is therefore assumed to have a negligible effect on the macroscopic flow. Instead, the fluid is treated as a continuous medium, enabling one to speak in terms of speed and density at a distinct point in space and time. This important assumption is known in fluid mechanics as the *continuum hypothesis*.

The system of equations that describes the motion of a fluid is known as the *Navier-Stokes equations*, independently derived by G.G. Stokes and M. Navier in the early 1800's. In the incompressible, viscous case the system consists of five equations: the continuity equation, three momentum equations and the

energy equation. These equations are presented here with a short outline of their origins; those interested in all underlying physics are referred to literature such as Anderson's *Fundamentals of aerodynamics* [1].

1. Continuity equation

   An obvious condition that holds for any fluid is that its mass is conserved. In an arbitrary control volume in space, the net mass flow through the boundaries should equal the rate of mass production within. In this report only non-production flow is considered so the latter is zero, yielding the following relation between the flow density ρ and speed $\boldsymbol{u}$:

   $$\rho_{,t} + (\rho u_\alpha)_{,\alpha} = 0. \tag{2.3}$$

2. Momentum equations

   Another important law of nature is conservation of momentum, known as Newton's second law: force equals rate of change of momentum. Forces exerted on a fluid fall in two categories. In terms of control volumes: *body forces* act on the entire volume (gravity etc., collected in $\boldsymbol{f}$) while *surface forces* act on its boundary (pressure p, shear stress τ_{ij}). The former is assumed to be known, the latter will be related to the flow field variables through constitutive relations. This gives three equations, one for each dimension i:

   $$(\rho u_i)_{,t} + (\rho u_i u_\alpha - \tau_{i\alpha})_{,\alpha} = \rho f_i. \tag{2.4}$$

   The following constitutive relations are used:

   - $\tau_{ij} = -p\delta_{ij} + \mu(u_{i,j} + u_{j,i} - \frac{2}{3}\delta_{ij}u_{k,k})$
   - $p = \rho RT$,

   where μ is the dynamic viscosity of the fluid, T the temperature and R the specific gas constant that connects the fluid's pressure, density and temperature. The δ_{ij} in the shear stress expression is the Kronecker delta, which is one of i equals j, and zero otherwise.

3. Energy equation

   For incompressible flows, where ρ is constant, the above system of four equations is complete. For compressible flows, however, an additional equation is needed for the extra unknown ρ. This is the energy equation, which represents the first law of thermodynamics: energy can neither be created nor destroyed, it can only change form. This means that the rate of change of the total (internal + kinetic) energy e should balance the work on the fluid and heat flux $\boldsymbol{q}$, yielding:

   $$(\rho e)_{,t} + (\rho u_\alpha e - \tau_{\alpha\beta}u_\beta + q_\alpha)_{,\alpha} = \rho f_\beta u_\beta, \tag{2.5}$$

   with again two constitutive relations to close the system:

   - $q_i = -\kappa T_{,i}$
   - $e = \frac{1}{2}u_\alpha u_\alpha + c_v T$.

Here κ is the thermal conductivity and c_v the specific heat at constant volume, related to the specific heat at constant pressure c_p via $c_p - c_v = R$.

In vector notation, the above system of five equations can be written compactly as follows:

$$\boldsymbol{U}_{,t} + \boldsymbol{F}_{\alpha,\alpha} = \boldsymbol{S}, \tag{2.6}$$

where $\boldsymbol{U}(\boldsymbol{x}, t) \in \mathbb{R}^5$ contains the so-called *conservation variables*, $\boldsymbol{F}_\alpha(\boldsymbol{x}, t) \in \mathbb{R}^5$ is the flux vector in the α-th direction and $\boldsymbol{S}(\boldsymbol{x}, t) \in \mathbb{R}^5$ a source vector. Along with the constitutive relations, (2.6) forms a closed, nonlinear system that completely describes the flow of a fluid.

The straightforward way to simulate a fluid flow would be to discretize a flow field and solve the above system numerically. This method is known as *Direct Numerical Simulation* (DNS). In practice, however, the smallest length scales are so small that the cost of such simulation is extremely high. This makes DNS unfeasible for realistic-size problems. Various models — as opposed to simulations — are developed to overcome this problem, such as *Reynolds Averaged Navier-Stokes* (RANS) and *Large Eddy Simulation* (LES). The model described in this chapter, the *Variational Multiscale* (VMS) method, is quite new. Because of its close relation to the better known LES model the following section first presents a brief overview of this model.

2.2 Large Eddy Simulation

Large Eddy Simulation (LES) is a numerical method that solves the system of Navier-Stokes equations on a much coarser grid than required for a direct numerical simulation. Since this grid obviously can not represent the smallest scales of fluid motion, a so called *eddy-viscosity model* is needed to compensate for this deficient resolution. Such a model tries to reconstruct the influences of the smallest, unresolved scales on the larger, resolved scales, using only those resolved scales.

An important idea behind LES is that of the *energy cascade*, introduced by Richardson [13] in 1920. The idea is that turbulence is composed of *eddies* of different sizes — an eddy being a certain localized turbulent motion. Richardson's notion is that the large eddies are unstable and break up, transferring their energy to somewhat smaller eddies, which in turn break up into smaller eddies. This energy cascade continues until at very small scale the eddy motion is stable, and kinetic energy is dissipated through molecular viscosity. Richardson summarized his paper as follows:

Big whorls have little whorls,
Which feed on their velocity;
And little whorls have lesser whorls,
And so on to viscosity.

Looking again at Da Vinci's drawing, Figure 2.2, it is clear that below a certain length scale the turbulent motions turn from anisotropic to isotropic, losing all information about the boundary imposed geometry of the large eddies. This divides the range of scales into two classes with markedly different properties. DNS resolves both of these, expending nearly all of its computational effort on the smallest scales of motion. LES resolves only the largest, exploiting the fact that the large eddies contain the bulk of the kinetic energy.

Pope [12] identifies four conceptual steps in LES:

1. A spatial filter is defined to decompose the velocity $\boldsymbol{u}$ into a filtered (resolved) component $\bar{\boldsymbol{u}}$ and a residual (unresolved) component $\hat{\boldsymbol{u}}$. The filtered velocity field $\bar{\boldsymbol{u}}$ represents the motion of the large eddies.
2. The filter is applied to the system of Navier-Stokes equations to derive a new system for the filtered velocity field $\bar{\boldsymbol{u}}$. Problems arise at the nonlinear terms, since those can not be expressed exclusively in terms of the filtered velocity $\bar{\boldsymbol{u}}$. This is known as the *closure problem*.
3. Closure is obtained by replacing the velocity $\boldsymbol{u}$ in the non-linear terms with the filtered velocity $\bar{\boldsymbol{u}}$ and correcting this with a model term: the closure.
4. The new system is solved for $\bar{\boldsymbol{u}}$, which provides an approximation of the large scale motions in the turbulent flow.

Although LES does give quite good results for certain flow problems and in fact is widely used, the method has its drawbacks. For instance at step 2, the filter operation does not commute with spatial differentiation on non-uniform grids, restricting the method to relatively simply geometries. Other problems arise at the walls, where the filter will either have to shrink or extend beyond the boundary. The major source of problems, however, is step 3. The closure term often fails to realistically model the influence of the neglected smallest eddies, in particular near the walls. Since this model applies to all large scale motions LES fails to converge to the true DNS solution, which in the worst case may lead to a result that is qualitatively different.

2.3 The Variational Multi-Scale method

The Variational Multi-Scale method (VMS), described a.o. by Hughes et al. [5], Collis [2] and Munts et al. [9], was designed to address some of the shortcomings of the previously described LES method. Both methods use a similar approach; the system of Navier-Stokes equations is solved on a grid that is much too coarse to represent the small scales of fluid motion, and a model is used to compensate for the inability to simulate the smallest eddies. Instead of a single decomposition in resolved and unresolved scales, however, VMS goes one step

further and identifies large and small resolved scales. This division is reflected by a corresponding division in function spaces:

$$\mathcal{V} = \bar{\mathcal{V}} \oplus \tilde{\mathcal{V}} \oplus \hat{\mathcal{V}}, \tag{2.7}$$

where $\mathcal{V}$ is the problem's solution space, containing space-time representations of the flow, and $\bar{\mathcal{V}}$, $\tilde{\mathcal{V}}$ and $\hat{\mathcal{V}}$ represent the large, small and unresolved scales, respectively. The first two are resolved and as such finite dimensional. For brevity the resolved space is denoted as:

$$\bar{\tilde{\mathcal{V}}} = \bar{\mathcal{V}} \oplus \tilde{\mathcal{V}}. \tag{2.8}$$

The system of Navier-Stokes equations was presented in strong form in Section 2.1 as Equation (2.6). Vectors $\boldsymbol{U}$ and $\boldsymbol{F}_\alpha$ can be expressed in any set of five flow field variables, joined in a solution vector $\boldsymbol{Y}(\boldsymbol{x}, t)$. This choice of variables defines the solution space $\mathcal{V}$. For practical reasons, it is chosen to define the flow field by density, speed and temperature:

$$\boldsymbol{Y} = \{\rho, u_1, u_2, u_3, T\}^T \in \mathcal{V}. \tag{2.9}$$

Using the inner product $(\boldsymbol{f}, \boldsymbol{g}) = \int_Q \boldsymbol{f}^T \boldsymbol{g} \, \mathrm{d}Q$, the inner product with an arbitrary test function $\boldsymbol{W} \in \mathcal{V}$ transforms Equation (2.6) to its weak form:

$$\int_Q \boldsymbol{W}^T (\boldsymbol{U}_{,t} + \boldsymbol{F}_{\alpha,\alpha}) \, \mathrm{d}Q = \int_Q \boldsymbol{W}^T \boldsymbol{S} \, \mathrm{d}Q \quad \forall \, \boldsymbol{W} \in \mathcal{V}, \tag{2.10}$$

or, expressed in $\boldsymbol{Y}$:

$$B(\boldsymbol{W}, \boldsymbol{Y}) = (\boldsymbol{W}, \boldsymbol{S}) \quad \forall \, \boldsymbol{W} \in \mathcal{V}, \tag{2.11}$$

for some operator $B : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$, that is linear in its first argument and non-linear in its second. The weak formulation is generally the first step of a solution procedure, such as a finite element method. Indeed, when the Navier-Stokes equations are reformulated in the VMS framework, this procedure will be continued in Section 2.4. The reason that the weak formulation is presented here already is that it is an essential part of VMS as well, because the test functions $\boldsymbol{W}$ are used to separate the large, small and unresolved scales. According to Equation (2.7), Equation (2.11) can be written as:

$$\begin{aligned} B(\bar{\boldsymbol{W}} + \tilde{\boldsymbol{W}} + \hat{\boldsymbol{W}}, \bar{\boldsymbol{Y}} + \tilde{\boldsymbol{Y}} + \hat{\boldsymbol{Y}}) = (\bar{\boldsymbol{W}} + \tilde{\boldsymbol{W}} + \hat{\boldsymbol{W}}, \boldsymbol{S}) \\ \forall \, \bar{\boldsymbol{W}} \in \bar{\mathcal{V}}, \tilde{\boldsymbol{W}} \in \tilde{\mathcal{V}}, \hat{\boldsymbol{W}} \in \hat{\mathcal{W}}. \end{aligned} \tag{2.12}$$

Taking only one element $\bar{\boldsymbol{W}}$, $\tilde{\boldsymbol{W}}$ or $\hat{\boldsymbol{W}}$ non-zero at a time, the following system of three equations is obtained:

$$\begin{aligned} B(\bar{\boldsymbol{W}}, \bar{\boldsymbol{Y}} + \tilde{\boldsymbol{Y}} + \hat{\boldsymbol{Y}}) &= (\bar{\boldsymbol{W}}, \boldsymbol{S}) \quad \forall \, \bar{\boldsymbol{W}} \in \bar{\mathcal{V}} \\ B(\tilde{\boldsymbol{W}}, \bar{\boldsymbol{Y}} + \tilde{\boldsymbol{Y}} + \hat{\boldsymbol{Y}}) &= (\tilde{\boldsymbol{W}}, \boldsymbol{S}) \quad \forall \, \tilde{\boldsymbol{W}} \in \tilde{\mathcal{V}} \\ B(\hat{\boldsymbol{W}}, \bar{\boldsymbol{Y}} + \tilde{\boldsymbol{Y}} + \hat{\boldsymbol{Y}}) &= (\hat{\boldsymbol{W}}, \boldsymbol{S}) \quad \forall \, \hat{\boldsymbol{W}} \in \hat{\mathcal{V}}. \end{aligned} \tag{2.13}$$

Equation (2.12) implies Equation (2.13) by construction. Since all terms in (2.13) are linear in their first argument, summing the three equations transforms the system back to (2.12). This proves the reverse implication, so Equations (2.13), (2.12) and hence (2.11) are all equivalent formulations of the original, exact Equation (2.6).

Reviewing the above procedure, the original system is reformulated in terms of unresolved, small and large scales by making a corresponding division in function spaces and writing the solution $\boldsymbol{Y}$ as a sum of its elements. The weak formulation acts as a kind of 'projection' of $\boldsymbol{Y}$ onto $\bar{\mathcal{V}}$, $\tilde{\mathcal{V}}$ and $\hat{\mathcal{V}}$. This is a striking difference with the LES method; the three scales are separated without the use of any filtering operation. As a consequence, the obtained formulation is still exact. Using test functions for the splitting eradicates all problems caused by the spatial filter (at the wall, on unstructured grids), without introducing a substitute operation as these test functions will be part of the solution procedure anyway.

The exactness of Equation (2.13) is lost when the unresolved scales $\hat{\boldsymbol{Y}}$ are ignored, which is necessary as these can not be represented on a coarse grid. The influence of these scales is present in both of the remaining large and small scale equations. However, according to Richardson's energy cascade introduced in Section 2.2, eddies influence mostly their nearby scales. Therefore, the influence of the unresolved scales on the largest scales is assumed negligible, so there the $\hat{\boldsymbol{Y}}$ term drops out. The unresolved scales dissipate energy from the small scales, so terms involving unresolved scales in the small scale equation will have to be modeled. Under these assumptions, the remaining nonlinear system can be written as:

$$\begin{aligned} \boldsymbol{B}(\bar{\boldsymbol{W}}, \bar{\boldsymbol{Y}} + \tilde{\boldsymbol{Y}}) &= (\bar{\boldsymbol{W}}, \boldsymbol{S}) \quad \forall\ \bar{\boldsymbol{W}} \in \bar{\mathcal{V}} \\ \boldsymbol{B}(\tilde{\boldsymbol{W}}, \bar{\boldsymbol{Y}} + \tilde{\boldsymbol{Y}}) + \boldsymbol{M}(\tilde{\boldsymbol{W}}, \tilde{\boldsymbol{Y}}) &= (\tilde{\boldsymbol{W}}, \boldsymbol{S}) \quad \forall\ \tilde{\boldsymbol{W}} \in \tilde{\mathcal{V}}, \end{aligned} \tag{2.14}$$

where $\boldsymbol{M}$ is the model term that acts on the small scales only. Summing the large and small scale equation yields the final, nonlinear system of equations:

$$\boldsymbol{B}(\bar{\tilde{\boldsymbol{W}}}, \bar{\tilde{\boldsymbol{Y}}}) + \boldsymbol{M}(\tilde{\boldsymbol{W}}, \tilde{\boldsymbol{Y}}) = (\bar{\tilde{\boldsymbol{W}}}, \boldsymbol{S}) \quad \forall\ \bar{\tilde{\boldsymbol{W}}} \in \bar{\tilde{\mathcal{V}}}. \tag{2.15}$$

Here $\bar{\tilde{\boldsymbol{Y}}}$ is the (finite dimensional) solution vector of the numerical scheme. In the following only finite dimensional approximations will be considered, and the distinction with the exact solution $\boldsymbol{Y}$ will be made no more. Therefore, from now on $\mathcal{V}$ will denote the solution space containing resolved scales, and its elements are $\boldsymbol{Y}$ and $\boldsymbol{W}$. The tilde does not change meaning; it will still denote the small components of the resolved scales. With these redefinitions, Equation (2.15) reads in integral form:

$$\int_Q \boldsymbol{W}^T(\boldsymbol{U}_{,t} + \boldsymbol{F}_{\alpha,\alpha})\ \mathrm{d}Q + \int_Q \tilde{\boldsymbol{W}}^T \tilde{\boldsymbol{F}}^m_{\alpha,\alpha}\ \mathrm{d}Q = \int_Q \boldsymbol{W}^T \boldsymbol{S}\ \mathrm{d}Q \quad \forall\ \boldsymbol{W} \in \mathcal{V}, \tag{2.16}$$

where $\tilde{\boldsymbol{F}}^m_\alpha$ is a model term. The tilde denotes its dependence on $\tilde{\boldsymbol{Y}}$. A final comparison with the exact Equation (2.10) gives a clear picture of the resulting VMS procedure: the unmodified Navier-Stokes equations are solved in weak

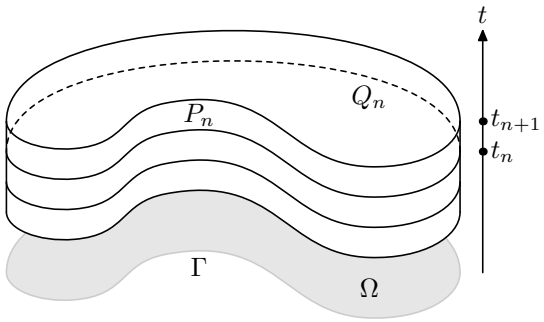

Figure 2.3: Schematic overview of the division of a space-time domain Q into successive time-slabs Q_n. Note that in reality, the spatial domain Ω is three-dimensional.

form on a coarse grid, and an extra term $\tilde{\boldsymbol{F}}_\alpha$ models the effect of the eddies that can not be resolved on this grid. Since this model term is a function of the small resolved scales only, the solution space $\mathcal{V}$ should be constructed such that it allows an easy separation of scales.

2.4 Construction of the solution space

Equation (2.16) will be solved using a finite element method. Therefore, first the solution space $\mathcal{V}$ has to be defined. Since $\boldsymbol{Y}$ represents a solution in both space and time, its domain Q is four-dimensional. In order to reduce the size of the final calculations this domain is first divided into a series of so called time-slabs Q_n, on which the restrictions $\boldsymbol{Y}|_{Q_n}$ can be calculated successively:

$$\begin{aligned} Q_n &= \Omega \times (t_n, t_{n+1}) \\ P_n &= \Gamma \times (t_n, t_{n+1}), \end{aligned} \tag{2.17}$$

where Ω denotes the spatial domain with boundary Γ and t_n and t_{n+1} are successive points in time. Figure 2.3 gives a schematic overview of this setup. Next, following standard finite element procedure, these time-slabs are subdivided into sub-domains $Q_{(n,i)}$ — the finite elements — such that $\cup_i Q_{(n,i)} = Q_n$ and $Q_{(n,i)} \cap Q_{(n,j)} = \varnothing$ if $i \neq j$. The solution space $\mathcal{V}$ can now defined via its restriction on these finite elements:

$$\mathcal{V} \ni \boldsymbol{Y}\Big|_{Q_{(n,i)}} = \sum_{a \in \mathcal{A}} N_{(n,i,a)} \boldsymbol{y}_{(n,i,a)}, \tag{2.18}$$

where $\mathcal{A}$ is an index set, $N_{(n,i,a)} : Q_{(n,i)} \to \mathbb{R}$ are polynomial basis functions and $\boldsymbol{y}_{(n,i,a)} \in \mathbb{R}^5$ are element vectors corresponding to time-slab n, element i, basis function a. The only restriction that is imposed on the element vectors is that the resulting function $\boldsymbol{Y}$ must be continuous within each time-slab. Continuity over successive time-slabs is not required, as shall be discussed later on.

The previous section pointed out that the function space $\mathcal{V}$ should allow easy separation of scales, because the model term in Equation (2.16) depends on

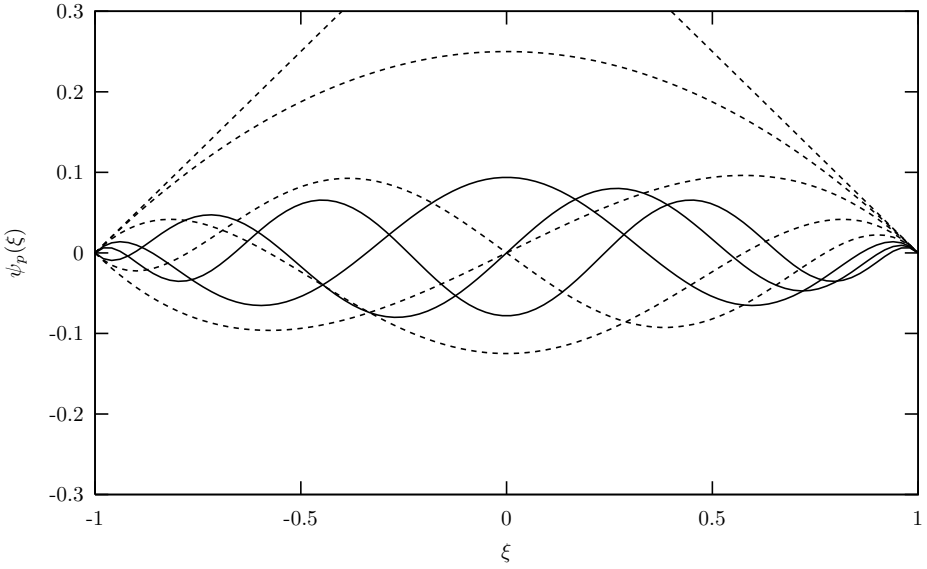

Figure 2.4: Polynomial Legendre expansion, corresponding to $\alpha = \beta = 0$ and $P = 8$ in (2.21). The solid lines correspond to the highest order modes $p = 5, 6, 7$. In this graph it is clear that high order modes represent small length scales.

the small scales $\tilde{\boldsymbol{Y}}$ only. Equation (2.18) suggests to choose the basis functions $N_{(n,i,a)}$ in such a way that each function can be identified with a certain scale; in that case the small-scale function space $\tilde{\mathcal{V}}$ can be defined as a sum over a subset of these basis functions:

$$\tilde{\mathcal{V}} \ni \tilde{\boldsymbol{Y}}\Big|_{Q_{(n,i)}} = \sum_{a \in \tilde{\mathcal{A}}} N_{(n,i,a)} \boldsymbol{y}_{(n,i,a)}, \tag{2.19}$$

where $\mathcal{A} \supset \tilde{\mathcal{A}}$ contains the indices of the basis functions that can be identified with small scales. Defined like this, $\tilde{\boldsymbol{Y}}$ and $\boldsymbol{Y}$ share the same element vector $\boldsymbol{y}_{(n,i,a)}$, which clearly makes it very easy to relate the one to the other in a numerical setting.

The construction of polynomial basis functions on a multi-dimensional domain is described in Chapter 3 of Karniadakis et al. [7]. Given a set of P polynomials $\psi_p : [-1, 1] \to \mathbb{R}$, the expansion on a four-dimensional domain is given by the tensor product:

$$\phi_{pqrs}(\xi_1, \xi_2, \xi_3, \xi_4) = \psi_p(\xi_1)\psi_q(\xi_2)\psi_r(\xi_3)\psi_s(\xi_4). \tag{2.20}$$

A set of P^4 basis functions is created for each element $Q_{(n,i)}$ by mapping ξ_1 – ξ_4 onto the finite element. For these basis functions to correspond to a range of scales, by construction the same must hold for the polynomial expansion set ψ_p from which they are derived. A *hierarchical modal p-type expansion* has this property, because its high order modes can be identified with small scales. This expansion is defined as follows:

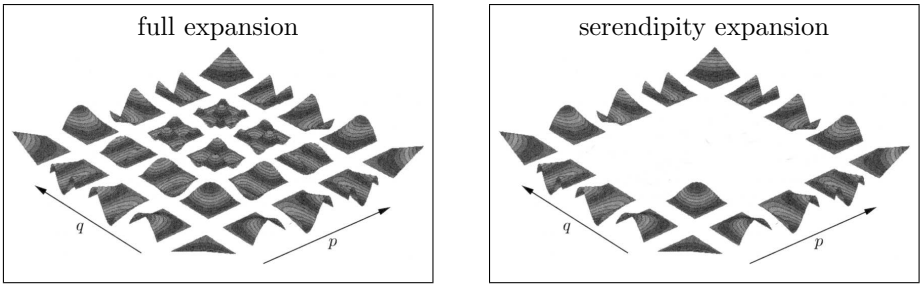

Figure 2.5: Two-dimensional modal p-type expansion of order $P = 4$, copied from Karniadakis et al. [7], Chapter 3. Left the full expansion, right the reduced serendipity expansion in which only 17 of the total 25 elements are retained. In more dimensions, the reduction is even more significant.

$$\psi_p(\xi) = \begin{cases} \frac{1-\xi}{2} & p = 0 \\ \frac{1-\xi}{2}\frac{1+\xi}{2} P_{p-1}^{\alpha,\beta}(\xi) & 0 < p < P \\ \frac{1+\xi}{2} & p = P, \end{cases} \tag{2.21}$$

where $P_p^{\alpha,\beta}(\xi)$ represents Jacobi polynomials of order p, for example Legendre ($\alpha = \beta = 0$) or Chebychev ($\alpha = \beta = -\frac{1}{2}$) polynomials. The scale separation property is clearly visible in Figure 2.4, which shows a Legendre expansion with the highest orders highlighted. This property is propagated via Equation (2.20) to the basis functions $N_{(n,i,a)}$.

An attractive side effect of the modal expansion is that, according to Karniadakis et al. [7], its hierarchical nature "permits the use of a reduced number of expansion modes as compared with those in the full tensor space", which is shown in Figure 2.5. The reduction of this so called *serendipity expansion* is especially significant in higher dimensions, like a four-dimensional space-time discretization. Consequently, this leads to a significant reduction in size of the final system of equations.

2.5 Discontinuous-Galerkin formulation

Now that the solution space $\mathcal{V}$ has been defined, the final system is obtained by substituting its elements $\boldsymbol{Y}$ and $\boldsymbol{W}$ into Equation (2.16), which is already in weak form due to the scale separating procedure applied in Section 2.3. Rewriting the left-hand terms using the divergence theorem yields:

$$\int_\Omega (\boldsymbol{W}^T|_{t_{n+1}^-}\boldsymbol{U}|_{t_{n+1}^-} - \boldsymbol{W}^T|_{t_n^+}\boldsymbol{U}|_{t_n^+})\,\mathrm{d}\Omega + \int_{P_n} (\boldsymbol{W}^T\boldsymbol{F}_\alpha + \tilde{\boldsymbol{W}}^T\tilde{\boldsymbol{F}}_\alpha^m)n_\alpha\,\mathrm{d}P$$
$$- \int_{Q_n} (\boldsymbol{W}_{,t}^T\boldsymbol{U} + \boldsymbol{W}_{,\alpha}^T\boldsymbol{F}_\alpha + \tilde{\boldsymbol{W}}_{,\alpha}\tilde{\boldsymbol{F}}_\alpha^m + \boldsymbol{W}^T\boldsymbol{S})\,\mathrm{d}Q = 0 \quad \forall\, \boldsymbol{W} \in \mathcal{V}, \tag{2.22}$$

where $\boldsymbol{n}$ in the second integral is the outward normal on P_n, $\boldsymbol{U}|_t$ is shorthand notation for $\boldsymbol{x} \to \boldsymbol{U}(\boldsymbol{x}, t)$ and t_n^+ and t_{n+1}^- are the initial and end time of time-

slab n, respectively. Note that $\boldsymbol{U}$ and $\boldsymbol{F}_\alpha$ are functions of $\boldsymbol{Y}$, and $\tilde{\boldsymbol{F}}_\alpha^m$ of $\tilde{\boldsymbol{Y}}$. The spatial boundary conditions enter through the second boundary integral, and the initial condition $\boldsymbol{U}|_{t_n^+}$ in the first integral is formed from the result from the preceding time-slab, $\boldsymbol{U}|_{t_n^-}$. This gives rise to the so called *jump condition*:

$$\int_\Omega \boldsymbol{W}^T|_{t_n^+}\boldsymbol{U}|_{t_n^+}\ \mathrm{d}\Omega = \int_\Omega \boldsymbol{W}^T|_{t_n^+}\boldsymbol{U}|_{t_n^-}\ \mathrm{d}\Omega \quad \forall\ \boldsymbol{W} \in \mathcal{V}, \tag{2.23}$$

where t_n^+ denotes the initial time in time-slab n and t_n^- the end time of the preceding slab $n-1$. This weakly enforced initial condition allows the solution to be discontinuous over the time-slabs, as noted in the previous section at the definition of $\mathcal{V}$. For this reason, this method is called a *time-discontinuous Galerkin* method.

The solution $\boldsymbol{Y}$ and test functions $\boldsymbol{W}$ are completely determined by their element vectors $\boldsymbol{y}_{(n,i,a)}$ and $\boldsymbol{w}_{(n,i,a)}$ via Equation (2.18). These element vectors are joined per time-slab in global vectors $\boldsymbol{y}_n$ and $\boldsymbol{w}_n$ of length N, from which they can be extracted by multiplication with a sparse, boolean matrix $\boldsymbol{M}_{(n,i,a)} \in \mathbb{R}^{N\times 5}$, called the *location operator*:

$$\begin{aligned} \boldsymbol{y}_{(n,i,a)} &= \boldsymbol{M}_{(n,i,a)}\boldsymbol{y}_n \\ \boldsymbol{w}_{(n,i,a)} &= \boldsymbol{M}_{(n,i,a)}\boldsymbol{w}_n. \end{aligned} \tag{2.24}$$

Location operators for adjacent elements may extract common values from $\boldsymbol{y}_n$ and $\boldsymbol{w}_n$ in order to satisfy the continuity condition. Note from Equation (2.21) that there are only two elements that have a non-zero value on the boundary: ψ_0 and ψ_P. This would suggest that the continuity constraint affects only a small subset of the final collection of basis functions. Regrettably, in multiple dimensions this does not hold, as many elements will be non-zero on the edges connecting two nodes. For instance in Equation (2.20), if $\psi_p \neq 0$ on the edge $\xi_1 = 1$, all combinations of ϕ_q, ϕ_r and ϕ_s will produce basis functions that have non-zero values on this edge.

Combining Equations (2.19) and (2.24), the test functions $\boldsymbol{W}$ can be expressed in terms of their global vectors $\boldsymbol{w}_n$:

$$\mathcal{V} \ni \boldsymbol{W}\Big|_{Q_{(n,i)}} = \left(\sum_{a\in\mathcal{A}} N_{(n,i,a)}\boldsymbol{M}_{(n,i,a)}\right)\boldsymbol{w}_n. \tag{2.25}$$

The small-scale functions $\tilde{\boldsymbol{W}}$ are identical, except that the polynomials are summed only over the subset $\tilde{\mathcal{A}}$. These expressions for $\boldsymbol{W}$ and $\tilde{\boldsymbol{W}}$ can now be substituted into the rewritten system of equations, Equation (2.22), but this requires the integrals to be split into subintegrals over the separate finite elements because the polynomials $N_{(n,i,a)}$ are defined per element $Q_{(n,i)}$ only. For brevity, these subintegrals will be denoted as vectors $\boldsymbol{v}_{(n,i,a)}$ and $\tilde{\boldsymbol{v}}_{(n,i,a)}$, the former dependent on $\boldsymbol{Y}$, the latter on $\tilde{\boldsymbol{Y}}$. When $\mathcal{I}_n$ denotes the index set of elements in time-slab n, Equation (2.22) transforms to:

$$\boldsymbol{w}_n^T \sum_{i\in\mathcal{I}_n} \left\{ \sum_{a\in\mathcal{A}} \boldsymbol{M}_{(n,i,a)}^T \boldsymbol{v}_{(n,i,a)} + \sum_{a\in\tilde{\mathcal{A}}} \boldsymbol{M}_{(n,i,a)}^T \tilde{\boldsymbol{v}}_{(n,i,a)} \right\} = 0 \quad \forall\ \boldsymbol{w}_n \in \mathbb{R}^N, \tag{2.26}$$

or, more compactly:

$$\boldsymbol{w}_n^T \boldsymbol{G}(\boldsymbol{y}_n, \boldsymbol{y}_{n-1}) = 0 \quad \forall\ \boldsymbol{w}_n \in \mathbb{R}^N, \tag{2.27}$$

where $\boldsymbol{G} : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^N$ is a non-linear operator, composed of all separate element integrals $\boldsymbol{v}_{(n,i,a)}$ and $\tilde{\boldsymbol{v}}_{(n,i,a)}$ that are put in place by the location operators $\boldsymbol{M}_{(n,i,a)}$. Its dependence on the previous time-slab solution $\boldsymbol{y}_{n-1}$ is due to the jump condition, Equation (2.23). With operator $\boldsymbol{G}$ defined, the requirement that $\boldsymbol{w}_n^T \boldsymbol{G}$ is zero for any vector $\boldsymbol{w}_n \in \mathbb{R}^N$ leads to the final, non-linear system of N equations:

$$\boldsymbol{G}(\boldsymbol{y}_n, \boldsymbol{y}_{n-1}) = \boldsymbol{0}. \tag{2.28}$$

For sake of completeness, the complete expressions of the subintegrals $\boldsymbol{v}_{(n,i,a)}$ and $\tilde{\boldsymbol{v}}_{(n,i,a)}$ are presented below. These will not be used further in this report.

$$\begin{aligned}
\boldsymbol{v}_{(n,i,a)} = &\int_{\Omega_{(n,i)}} \left(N_{(n,i,a)}|_{t_{n+1}^-} \boldsymbol{U}|_{t_{n+1}^-} - N_{(n,i,a)}|_{t_n^+} \boldsymbol{U}|_{t_n^-} \right) \mathrm{d}\Omega \\
&+ \int_{P_{(n,i)}} N_{(n,i,a)} \boldsymbol{F}_\alpha n_\alpha \ \mathrm{d}P \\
&- \int_{Q_{(n,i)}} \left(N_{(n,i,a),t} \boldsymbol{U} + N_{(n,i,a),\alpha} \boldsymbol{F}_\alpha + N_{(n,i,a)} \boldsymbol{S} \right) \mathrm{d}Q,
\end{aligned}$$

$$\begin{aligned}
\tilde{\boldsymbol{v}}_{(n,i,a)} = &\int_{P_{(n,i)}} N_{(n,i,a)} \tilde{\boldsymbol{F}}_\alpha^m n_\alpha \ \mathrm{d}P \\
&- \int_{Q_{(n,i)}} N_{(n,i,a),\alpha} \tilde{\boldsymbol{F}}_\alpha^m \ \mathrm{d}Q.
\end{aligned}$$

2.6 Linearization

Non-linear equations such as Equation (2.28) are generally solved with an iterative solution method. Such an iterative method starts with an initial guess $\boldsymbol{y}_n^0$ and uses a recurrence relation to construct a sequence of approximations $\boldsymbol{y}_n^k$ that converges to the exact solution $\boldsymbol{y}_n$, assuming that this solution is unique. The iterations can be stopped when $\boldsymbol{y}_n^k$ is considered a good enough approximation of $\boldsymbol{y}_n$:

$$\|G(\boldsymbol{y}_n^k, \boldsymbol{y}_{n-1})\| < \epsilon, \tag{2.29}$$

where $\|\cdot\|$ is an appropriately chosen norm and ϵ is a measure of the desired precision. The solution on the preceding time-slab $\boldsymbol{y}_{n-1}$ is known. It seems like an obvious choice to use this previous solution as a starting vector $\boldsymbol{y}_n^0$ for the new iterations. Note, however, that the jump condition from Section 2.5 does not require meshes on successive time-slabs to correspond in any way. Even if $\boldsymbol{y}_{n-1}$ has the same length as $\boldsymbol{y}_n$, it can represent a completely different flow on both

time-slabs. This is for example a point of concern for VMS implementations that deal with time-changing domains.

A widely used root-finding algorithm is *Newton's method*, which is based on a Taylor expansion. For a scalar function $f : \mathbb{R} \to \mathbb{R}$, the first two terms of this expansion read:

$$f(x + h) = f(x) + hf'(x) + \mathcal{O}(h^2), \tag{2.30}$$

where $\mathcal{O}$ is Landau's order symbol. If $x + h$ is a root of f, and x is an approximation of this root, the error h will be small. Under these assumptions, the above expression can be rewritten as:

$$h \approx -\frac{f(x)}{f'(x)}. \tag{2.31}$$

Here h is an approximation of the error, because the high order terms of the Taylor expansion are neglected. Therefore, adding this error to an approximate solution will not result in an exact root of f. It will, however, result in a much improved approximation of this root. Each time that this procedure is repeated the error decreases, ever more justifying the neglection of high order terms. The resulting, rapidly converging algorithm is described by the following recurrence relation:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{2.32}$$

This recurrence depends only on the most recent approximation x_k, so previous approximations do not have to be kept in memory. This short recurrence property becomes important when Newton's method is generalized to vector functions, which arguments can be very large. For such functions the derivative is replaced with a Jacobian matrix. Applied to Equation (2.28), Newton's method reads:

$$\boldsymbol{y}_n^{k+1} = \boldsymbol{y}_n^k - \left(\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{y}_n}(\boldsymbol{y}_n^k, \boldsymbol{y}_{n-1})\right)^{-1} \boldsymbol{G}(\boldsymbol{y}_n^k, \boldsymbol{y}_{n-1}), \tag{2.33}$$

where $\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{y}_n} \in \mathbb{R}^{N \times N}$ is the Jacobian matrix of $\boldsymbol{G}$. Recall that $\boldsymbol{G}$ is very large, containing equations for every degree of freedom on the four-dimensional grid. And even though its Jacobian will be a sparse matrix, its inverse will not. This has major consequences on both memory usage and computing time, which makes it impossible to use the Newton iterations in the above form. Instead, $\boldsymbol{y}_n^{k+1}$ will be calculated as follows:

$$\boldsymbol{y}_n^{k+1} = \boldsymbol{y}_n^k - \boldsymbol{x}, \tag{2.34}$$

where $\boldsymbol{x}$ solves

$$\left(\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{y}_n}(\boldsymbol{y}_n^k, \boldsymbol{y}_{n-1})\right) \boldsymbol{x} = \boldsymbol{G}(\boldsymbol{y}_n^k, \boldsymbol{y}_{n-1}). \tag{2.35}$$

This is a linear system of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A}$ and $\boldsymbol{b}$ change with every iteration. Solving this system with a direct method is infeasible for the same reasons that impede the direct inversion of the Jacobian. Besides, apart from these practical problems it is a clear waste of computing time to solve $\boldsymbol{y}_n^k$ with machine precision, since it is only an intermediate step in Newton's method. An

iterative solution procedure within each Newton iteration is the natural choice, since it calculates $\boldsymbol{y}_n^k$ with controllable precision, while exploiting the sparsity of the Jacobian matrix.

For linear systems, a large number of iterative solvers are available. However, many of those apply only if the matrix has certain properties, like symmetry, or they are known to work especially well or to break down under certain conditions. To find out which solvers apply to the problem under consideration, more information is needed on the Jacobian of $\boldsymbol{G}$. By Equations (2.26) and (2.27), the Jacobian matrix can be written as:

$$\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{y}_n} = \sum_{i \in \mathcal{I}_n} \Big\{ \sum_{a \in \mathcal{A}} \boldsymbol{M}_{(n,i,a)}^T \frac{\partial \boldsymbol{v}_{(n,i,a)}}{\partial \boldsymbol{y}_n} + \sum_{a \in \tilde{\mathcal{A}}} \boldsymbol{M}_{(n,i,a)}^T \frac{\partial \tilde{\boldsymbol{v}}_{(n,i,a)}}{\partial \boldsymbol{y}_n} \Big\}, \tag{2.36}$$

where $\frac{\partial \boldsymbol{v}_{(n,i,a)}}{\partial \boldsymbol{y}_n} \in \mathbb{R}^{5 \times N}$ and $\frac{\partial \tilde{\boldsymbol{v}}_{(n,i,a)}}{\partial \boldsymbol{y}_n} \in \mathbb{R}^{5 \times N}$ are the Jacobian matrices of $\boldsymbol{v}_{(n,i,a)}$ and $\tilde{\boldsymbol{v}}_{(n,i,a)}$, respectively. Since $\boldsymbol{v}_{(n,i,a)}$ and $\tilde{\boldsymbol{v}}_{(n,i,a)}$ consist of integrals over a single finite element i, only a subset of $\boldsymbol{y}_n$ — the element vectors $\boldsymbol{y}_{(n,i,b)}$ — produces non-zero derivatives. Hence, the Jacobian of $\boldsymbol{v}_{(n,i,a)}$ can be written as:

$$\frac{\partial \boldsymbol{v}_{(n,i,a)}}{\partial \boldsymbol{y}_n} = \sum_{b \in \mathcal{A}} \frac{\partial \boldsymbol{v}_{(n,i,a)}}{\partial \boldsymbol{y}_{(n,i,b)}} \boldsymbol{M}_{(n,i,b)}, \tag{2.37}$$

where $\frac{\partial \boldsymbol{v}_{(n,i,a)}}{\partial \boldsymbol{y}_{(n,i,b)}} \in \mathbb{R}^{5 \times 5}$ is a subset of the full Jacobian matrix, corresponding to the derivatives of $\boldsymbol{y}_{(n,i,b)}$. The location operator moves its five columns to their positions in the full Jacobian matrix. A similar expression holds for $\tilde{\boldsymbol{v}}_{(n,i,a)}$, where b can even be restricted to $\tilde{\mathcal{A}}$ because $\tilde{\boldsymbol{v}}_{(n,i,a)}$ depends only on the small scales $\tilde{\boldsymbol{Y}}$. Substitution into (2.36) yields a new expression for the Jacobian matrix:

$$\begin{aligned} \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{y}_n}(\boldsymbol{y}_n, \boldsymbol{y}_{n-1}) = \sum_{i \in \mathcal{I}_n} \Big\{ & \sum_{(a,b) \in \mathcal{A}^2} \boldsymbol{M}_{(n,i,a)}^T \frac{\partial \boldsymbol{v}_{(n,i,a)}}{\partial \boldsymbol{y}_{(n,i,b)}} \boldsymbol{M}_{(n,i,b)} \\ & + \sum_{(a,b) \in \tilde{\mathcal{A}}^2} \boldsymbol{M}_{(n,i,a)}^T \frac{\partial \tilde{\boldsymbol{v}}_{(n,i,a)}}{\partial \boldsymbol{y}_{(n,i,b)}} \boldsymbol{M}_{(n,i,b)} \Big\} \end{aligned} \tag{2.38}$$

Reviewing this expression, it is clear that if $\frac{\partial \boldsymbol{v}_{(n,i,a)}}{\partial \boldsymbol{y}_{(n,i,b)}} = \left(\frac{\partial \boldsymbol{v}_{(n,i,b)}}{\partial \boldsymbol{y}_{(n,i,a)}} \right)^T$, and idem for $\tilde{\boldsymbol{v}}_{(n,i,a)}$, then the Jacobian matrix is symmetric. This is not the case. What does follow from this observation is that the sparsity pattern is symmetric. The matrix itself, however, is not. This excludes a lot of solution methods from the list of candidates, like the popular Conjugate Gradient method. The methods that remain will be reviewed in the next chapter.

Chapter 3

Iterative solution methods

When faced with a large system of equations, a possible approach to solve it is to make an initial guess and improve it, time and again, until it becomes a satisfactory approximation of the exact solution. This is the approach used by iterative solution methods, which form an important area of numerical mathematics. Many different iterative methods have been developed, designed to perform well (often exclusively) under certain conditions, or to have other favourable properties such as a large degree of parallelization.

This chapter gives an overview of the most important iterative methods available for general, non-symmetric matrices. Section 3.1 describes the class of basic iterative methods and Section 3.2 that of the more advanced Krylov subspace methods. Most information for these sections is obtained from Yousef Saad's *Iterative Methods for Sparse Linear Systems* [14]. Section 3.3 gives an overview of preconditioning techniques aim to accelerate convergence by modifying the coefficient matrix, and finally Section 3.4 shows how the same can be achieved by running some calculations in parallel on multiple machines.

3.1 Basic iterative methods

An equation of the form $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{A}$ is an $N \times N$ coefficient matrix, $\boldsymbol{b}$ a right-hand side vector and $\boldsymbol{x}$ a vector of unknowns, represents a system of N equations:

$$\begin{aligned} a_{1,1}x_1 + \cdots + a_{1,N}x_N &= b_1 \\ &\vdots \\ a_{N,1}x_1 + \cdots + a_{N,N}x_N &= b_N. \end{aligned}$$

Several direct methods exist to solve this system. Mostly used is *Gaussian elimination*. This method solves the first equation for x_1 and substitutes the

solution into the following equations, then solves the second equation for x_2, and so on until at the N-th equation the value of x_N is obtained. Then this value is substituted back into the previous equations, yielding x_{N-1}, x_{N-2} and so on, down to x_1. This procedure has some serious drawbacks:

1. It may break down, if no preventive measures are taken. For instance if $a_{1,1} = 0$, the first equation can not be solved for x_1 and the methods fails, whether the system has a solution or not. This can be fixed by pivoting the equations, reordering them in such a way that the procedure does not break down. This is always possible, but is does require extra work that should be accounted for.
2. Floating point errors can accumulate, with detrimental effects on the final solution. This can occur even for well conditioned matrices. This too can be remedied through pivoting.
3. For sparse matrices, the required amount of memory can be considerably larger than needed to store $\boldsymbol{A}$. In many applications, many of the coefficients $a_{i,j}$ are zero and thus do not have to be stored. However, during the Gaussian process, many of these elements will become non zero. Again, pivoting can reduce the amount of fill-in, but it may still be significant.

For large, sparse systems, iterative methods are often better suited. These methods repeatedly improve an approximate solution. The coefficient matrix $\boldsymbol{A}$ is used only in the context of matrix-vector multiplication so its sparsity can be fully exploited, which is beneficial for both work and memory. For problems in three or more dimensions, the amount of work is often — though not always — considerably less then that of direct methods, because the iterations can be stopped as soon as the error is considered small enough. Often it does not make sense to calculate $\boldsymbol{x}$ beyond a certain accuracy, as matrix $\boldsymbol{A}$ is itself the result of an approximation. With direct methods this is unavoidable since no approximate solution can be formed during the process.

Starting with an initial guess $(x_1, \ldots, x_N)^T$, a straightforward iterative solution procedure is to solve each equation for one of the unknowns, keeping the other unknowns at their old value. When the first equation is solved for x_1, the second for x_2, and so on, the following iterative scheme is obtained:

$$\begin{aligned} x_1^{(m+1)} &= \frac{b_1}{a_{1,1}} - \frac{a_{1,2}}{a_{1,1}} x_2^{(m)} - \cdots - \frac{a_{1,N}}{a_{1,1}} x_N^{(m)} \\ &\vdots \\ x_N^{(m+1)} &= \frac{b_N}{a_{N,1}} - \frac{a_{N,1}}{a_{N,1}} x_1^{(m)} - \cdots - \frac{a_{N,N-1}}{a_{N,1}} x_{N-1}^{(m)}. \end{aligned}$$

When $\boldsymbol{M}$ is a diagonal matrix with the diagonal elements of $\boldsymbol{A}$, then this scheme can be written in vector notation as $\boldsymbol{x}_{m+1} = \boldsymbol{M}^{-1}(\boldsymbol{b} - (\boldsymbol{A} - \boldsymbol{M})\boldsymbol{x}_m)$. The component $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_m$ is called the *residual vector*, which is a measure of the closeness of $\boldsymbol{x}_m$ to the exact solution $\boldsymbol{x}$ as can be seen by writing it as $\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}_m)$. Denoting this residual by $\boldsymbol{r}_m$, the iterative scheme simplifies to:

$$\boldsymbol{x}_{m+1} = \boldsymbol{x}_m + \boldsymbol{M}^{-1}\boldsymbol{r}_m. \tag{3.1}$$

This is the general form of a *basic iterative method* (BIM). Various other choices for the *iteration matrix* $\boldsymbol{M}$ are possible. The above case where $\boldsymbol{M}$ is the diagonal matrix of $\boldsymbol{A}$ is known as the *Gauss Jacobi iteration*. Another option is to use the lower diagonal part of $\boldsymbol{A}$, including the diagonal. This results in the *Gauss Seidel iteration*, which corresponds to solving each equation with the most recently calculated values of x_i. This generally leads to a much better convergence rate.

Convergence of any of these methods, however, can be guaranteed only under certain conditions. Note that if $\boldsymbol{M}$ equals $\boldsymbol{A}$, the procedure converges to the exact solution in a single iteration: $\boldsymbol{x}_{m+1} = \boldsymbol{x}_m + \boldsymbol{A}^{-1}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_m) = \boldsymbol{A}^{-1}\boldsymbol{b}$. Generally speaking, a basic iterative method converges if matrix $\boldsymbol{M}$ is close enough to $\boldsymbol{A}$ in some sense. This is defined as follows. Let the error of an approximation be denoted as $\boldsymbol{e}_m = \boldsymbol{x} - \boldsymbol{x}_m$. With this notation, the residual becomes $\boldsymbol{r}_m = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_m = \boldsymbol{A}\boldsymbol{e}_m$. Substitution in Equation (3.1) yields:

$$\boldsymbol{e}_{m+1} = (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{e}_m = (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})^{m+1}\boldsymbol{e}_0. \tag{3.2}$$

Clearly, the error converges to zero if all eigenvalues of $\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}$ are smaller than one in absolute value. It follows that the convergence requirements for $\boldsymbol{A}$ depend on the iteration matrix $\boldsymbol{M}$, hence on the iterative method used. For Gauss Jacobi, for instance, a sufficient condition is that $\boldsymbol{A}$ is strictly diagonal dominant.

From Equation (3.2) it also follows that the convergence rate depends on the largest eigenvalue, or spectral radius, of $\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}$. To exploit this, the *Successive Over-Relaxation* (SOR) method is introduced. This is a variant of Gauss Seidel, in which the iterates are a linear combination of old and new values. In matrix notation this corresponds to scaling the off-diagonal elements of $\boldsymbol{M}$ with a factor ω. The optimal value of ω is the minimizer of the spectral radius $\rho(\boldsymbol{I} - \boldsymbol{M}_\omega^{-1}\boldsymbol{A})$. Since this involves eigenvalue calculations, the exact optimum is hard to determine in practice, but estimation schemes do exist.

SOR is a generalization of Gauss Seidel, and therefore in general the fastest of the three iterative methods mentioned. However, its convergence properties are still much worse than that of the more advanced methods that will be described next. Nowadays, basic iterative methods are seldom used the way they are presented here. They still have a role to play though: Section 3.3 shows how these methods have found a new use as preconditioners.

3.2 Krylov subspace methods

Krylov subspace methods, like basic iterative methods, are solution methods that solve a system of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ by iteratively improving an approximate solution $\boldsymbol{x}_m$. The main difference is the way in which they are derived. Krylov subspace methods are based on the idea that the inverse of a matrix $\boldsymbol{A}$ can be approximated by a polynomial of $\boldsymbol{A}$:

$$\boldsymbol{A}^{-1} \approx p(\boldsymbol{A}). \tag{3.3}$$

The approximate solutions follow from $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b} \approx p(\boldsymbol{A})\boldsymbol{b}$. Since often a good initial guess $\boldsymbol{x}_0$ is available, the system is first rewritten as $\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}_0) = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0 = \boldsymbol{r}_0$. Then, when p_m denotes a polynomial of maximum order $m-1$, the iterates of a Krylov subspace method can be written as:

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + p_m(\boldsymbol{A})\boldsymbol{r}_0. \tag{3.4}$$

The construction of a polynomial p_m of order less than m corresponds to finding an element $\boldsymbol{k}$ from an m-dimensional subspace $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0)$ such that $\boldsymbol{x}_m = \boldsymbol{x}_0 + \boldsymbol{k} \approx \boldsymbol{x}$, where $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0)$ is the m-th order Krylov subspace defined as:

$$\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0) = \text{span}\{\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \boldsymbol{A}^2\boldsymbol{r}_0, \ldots, \boldsymbol{A}^{m-1}\boldsymbol{r}_0\}. \tag{3.5}$$

When no ambiguity is possible, the Krylov subspaces will be denoted simply as $\mathcal{K}_m$. By construction, the m-th subspace is m-dimensional, meaning that it can be spanned by m basis vectors $\boldsymbol{v}_i$. When these basis vectors are joined in a matrix $\boldsymbol{V}_m = \begin{pmatrix}\boldsymbol{v}_1 & \ldots & \boldsymbol{v}_m\end{pmatrix}$, each Krylov subspace element $\boldsymbol{k} \in \mathcal{K}_m$ can be mapped to a vector $\boldsymbol{y} \in \mathbb{R}^m$ via $\boldsymbol{k} = \boldsymbol{V}_m\boldsymbol{y}$. Hence, given a basis $\boldsymbol{V}_m$, the iterates of any Krylov subspace method can be written as:

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + \boldsymbol{V}_m\boldsymbol{y}_m, \tag{3.6}$$

and its residuals $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_m$ as:

$$\boldsymbol{r}_m = \boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{V}_m\boldsymbol{y}_m, \tag{3.7}$$

where $\boldsymbol{y}_m$ is a real-valued, m-element vector. Many methods for finding an optimal or near-optimal $\boldsymbol{y}_m$ are developed, collectively called Krylov subspace methods. A common element of these methods is the construction of useful set of basis vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$ for $\mathcal{K}_m$. The basis suggested by (3.5), $\boldsymbol{v}_i = \boldsymbol{A}^i\boldsymbol{r}_0$, can easily be expanded from one Krylov subspace to the next by applying $\boldsymbol{A}$ to the 'latest' vector $\boldsymbol{v}_m$, and adding the result to the basis. The following Theorem states that this property is not specific for this particular set of bases, but holds for any increasing set of basis vectors:

Theorem 1. *Let $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots$ be a collection of basis vectors such that $\mathcal{K}_m = \text{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\}$ for all m. Then $\mathcal{K}_{m+1} = \mathcal{K}_m \oplus \text{span}\{\boldsymbol{A}\boldsymbol{v}_m\}$.*

Proof: this can be shown by induction. The first basis vector $\boldsymbol{v}_1$ is necessarily a multiple of $\boldsymbol{r}_0$, so the theorem is clearly true for $m = 1$. From (3.5) it follows that $\mathcal{K}_{m+1} = \mathcal{K}_1 \oplus \boldsymbol{A}\mathcal{K}_m$, where $\boldsymbol{A}\mathcal{K}_m = \{\boldsymbol{A}\boldsymbol{v} \;\forall\; \boldsymbol{v} \in \mathcal{K}_m\}$. Given that the theorem is true for all $m < j$, repeated application of the theorem shows its validity for $m = j$:

$$\begin{aligned}
\mathcal{K}_{j+1} &= \mathcal{K}_1 \oplus \boldsymbol{A}\mathcal{K}_j \\
&= \mathcal{K}_1 \oplus \text{span}\{\boldsymbol{A}\boldsymbol{v}_1, \ldots \boldsymbol{A}\boldsymbol{v}_j\} \\
&= \mathcal{K}_2 \oplus \text{span}\{\boldsymbol{A}\boldsymbol{v}_2, \ldots \boldsymbol{A}\boldsymbol{v}_j\} \\
&= \ldots \text{ since } K_m \oplus \text{span}\{\boldsymbol{A}\boldsymbol{v}_m\} = K_{m+1} \;\forall\; m < j \\
&= \mathcal{K}_j \oplus \text{span}\{\boldsymbol{A}\boldsymbol{v}_j\}
\end{aligned}$$

□

3.2.1 Full Orthogonalization Method

Theorem 1 implies a large freedom in constructing a basis for $\mathcal{K}_m$. One of such constructions is *Arnoldi's method*, which uses an orthogonal projection to make each newly calculated vector $\boldsymbol{A}\boldsymbol{v}_m$ perpendicular to the basis vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$. The result is normalized. This procedure corresponds to solving $\boldsymbol{v}_{m+1}$ from:

$$\boldsymbol{A}\boldsymbol{v}_m = h_{1,m}\boldsymbol{v}_1 + \cdots + h_{m,m}\boldsymbol{v}_m + h_{m+1,m}\boldsymbol{v}_{m+1}, \quad \boldsymbol{v}_1 \perp \ldots \perp \boldsymbol{v}_{m+1}. \tag{3.8}$$

The coefficients can be calculated sequentially: $h_{i,m} = \boldsymbol{v}_i^T \boldsymbol{A}\boldsymbol{v}_m$ for each $i \leq m$ and $h_{m+1,m} = \|\boldsymbol{v}_{m+1}\|_2^{-1}$ (lastly) to normalize the remaining vector. This procedure breaks down if $\boldsymbol{v}_{m+1} = \boldsymbol{0}$, but it will be shown that this is not a problem. The new basis vector is orthonormal to all previously calculated basis vectors. Since the span has not changed, the result is an orthonormal basis for $\mathcal{K}_{m+1}$. When the first m basis vectors are joined in a matrix $\boldsymbol{V}_m$, the first m Arnoldi iterations can be written as:

$$\boldsymbol{A}\boldsymbol{V}_m = \boldsymbol{V}_m\boldsymbol{H}_m + h_{m+1,m}\boldsymbol{v}_{m+1}\boldsymbol{e}_m^T, \tag{3.9}$$

where $\boldsymbol{e}_m$ is the last unit vector in $\mathbb{R}^m$ and $\boldsymbol{H}_m$ is the $m \times m$ Hessenberg matrix built of the coefficients $h_{i,j}$:

$$\boldsymbol{H}_m = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,m} \\ h_{2,1} & & & \vdots \\ & \ddots & & \vdots \\ 0 & & h_{m,m-1} & h_{m,m} \end{pmatrix}. \tag{3.10}$$

It was noted earlier that the first basis vector $\boldsymbol{v}_1$ is a multiple of $\boldsymbol{r}_0$ in every basis, so $\boldsymbol{r}_0$ can be written as $\beta\boldsymbol{v}_1 = \beta\boldsymbol{V}_m\boldsymbol{e}_1$, where $\boldsymbol{e}_1$ is the first unit vector in $\mathbb{R}^m$ and β a real constant. Together with the expression for $\boldsymbol{A}\boldsymbol{V}_m$ from Equation (3.9), the residual Equation (3.7) transforms into:

$$\boldsymbol{r}_m = \boldsymbol{V}_m(\beta\boldsymbol{e}_1 - \boldsymbol{H}_m\boldsymbol{y}_m) - \boldsymbol{v}_{m+1}(\boldsymbol{e}_m^T\boldsymbol{y}_m). \tag{3.11}$$

When the Arnoldi procedure breaks down due to a zero basis vector $\boldsymbol{v}_{m+1}$, this expression reduces to $\boldsymbol{r}_m = \boldsymbol{V}_m(\beta\boldsymbol{e}_1 - \boldsymbol{H}_m\boldsymbol{y}_m)$. Matrix $\boldsymbol{H}_m$ is non singular, which means that there exists a vector $\boldsymbol{y}_m$ for which the residual is zero — in other words, a solution is part of the Krylov subspace. Consequently, Arnoldi breaks down only when the exact solution can be found. This property is called 'lucky breakdown'.

A Krylov subspace method that is based on Arnoldi's procedure is the *Full Orthogonalization Method* (FOM). This method chooses $\boldsymbol{y}_m \in \mathbb{R}^m$ in such a way that the corresponding residual $\boldsymbol{r}_m$ is perpendicular to each element in the Krylov subspace $\mathcal{K}_m$, so $\boldsymbol{V}_m^T\boldsymbol{r}_m = \boldsymbol{0}$. Since the basis vectors are orthonormal, it follows directly from (3.11) that

$$\boldsymbol{H}_m\boldsymbol{y}_m = \beta\boldsymbol{e}_1 \tag{3.12}$$

and consequently:

$$\boldsymbol{r}_m = \sigma_m\boldsymbol{v}_{m+1}, \tag{3.13}$$

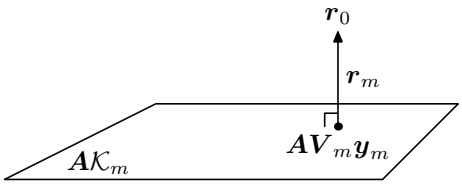

Figure 3.1: Schematic picture of equation (3.14) from the GMRES method. Solution $\boldsymbol{y}_m$ minimizes the residual $\boldsymbol{r}_m = \boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{V}_m\boldsymbol{y}_m$, which corresponds to making $\boldsymbol{r}_m$ perpendicular to $\boldsymbol{A}\mathcal{K}_m$.

where $\sigma_m = -\boldsymbol{e}_m^T\boldsymbol{y}_m$. This confirms that the exact solution is found the moment that Arnoldi breaks down. At any time during the orthonormalization process, an iterate $\boldsymbol{x}_m$ can be constructed by solving Equation (3.12) for $\boldsymbol{y}_m$ and substituting the solution into Equation (3.6). Efficient solution procedures are possible due to the Hessenberg structure of matrix $\boldsymbol{H}_m$. However, explicit calculation of $\boldsymbol{x}_m$ during the iterative process is not necessary. Convergence can be measured by the size of the residual which, according to Equation (3.13), is simply the last element of $\boldsymbol{y}_m$.

3.2.2 Generalized Minimum Residual Method

Another Krylov subspace method that is based on Arnoldi orthonormalization is the *Generalized Minimum Residual Method* (GMRES), which defines $\boldsymbol{y}_m$ as the minimizer of the residual $\boldsymbol{r}_m$, Equation (3.7), measured in the the 2-norm:

$$\boldsymbol{y}_m = \underset{\boldsymbol{y}\in\mathbb{R}^m}{\operatorname{argmin}} \|\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{V}_m\boldsymbol{y}\|_2. \tag{3.14}$$

This is a least squares problem; $\boldsymbol{y}_m$ is the best possible solution for the overdetermined system $\boldsymbol{A}\boldsymbol{V}_m\boldsymbol{y} = \boldsymbol{r}_0$. From linear least squares theory it is known that the solution is optimal if $\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{V}_m\boldsymbol{y}_m$ is perpendicular the the space spanned by $\boldsymbol{A}\boldsymbol{V}_m\boldsymbol{y}$ for all $\boldsymbol{y} \in \mathbb{R}^m$, which is $\boldsymbol{A}\mathcal{K}_m$. See also figure 3.1. Therefore, GMRES is very similar to FOM, the only difference being that GMRES makes $\boldsymbol{r}_m$ perpendicular to $\boldsymbol{A}\mathcal{K}_m$ instead of $\mathcal{K}_m$.

It is convenient to reformulate Equation (3.9) as:

$$\boldsymbol{A}\boldsymbol{V}_m = \boldsymbol{V}_{m+1}\bar{\boldsymbol{H}}_m, \tag{3.15}$$

where $\bar{\boldsymbol{H}}_m$ is an $(m+1)\times m$ matrix that consists of matrix $\boldsymbol{H}_m$ and an additional row, containing the coefficient $h_{m+1,m}$:

$$\bar{\boldsymbol{H}}_m = \begin{pmatrix} h_{1,1} & \dots & h_{1,m} \\ h_{2,1} & & \vdots \\ & \ddots & \vdots \\ 0 & & h_{m+1,m} \end{pmatrix}. \tag{3.16}$$

Analogous to Equation (3.11), the residual can now be written as:

$$\boldsymbol{r}_m = \boldsymbol{V}_{m+1}(\beta\boldsymbol{e}_1 - \bar{\boldsymbol{H}}_m\boldsymbol{y}_m). \tag{3.17}$$

The orthogonality condition thus becomes $(\boldsymbol{A}\boldsymbol{V}_m)^T\boldsymbol{r}_m = \bar{\boldsymbol{H}}_m^T\boldsymbol{V}_{m+1}^T\boldsymbol{r}_m = \boldsymbol{0}$. Since $\boldsymbol{V}_{m+1}^T\boldsymbol{V}_{m+1}$ equals identity, the GMRES analogue to Equation (3.12) is:

$$(\bar{\boldsymbol{H}}_m^T\bar{\boldsymbol{H}}_m)\boldsymbol{y}_m = \beta\bar{\boldsymbol{H}}_m^T\boldsymbol{e}_1. \tag{3.18}$$

In practice, however, $\boldsymbol{y}_m$ is not computed this way. Instead, a series of matrix transformations called *Givens rotations* is applied during the GMRES method in order to make $\bar{\boldsymbol{H}}_m$ triangular, thus transforming the least-squares problem corresponding to Equation (3.14) into a triangular system. The residual is obtained 'for free' in the process, so this solution $\boldsymbol{y}$ does not even have to be calculated until after the iterations. The full details of this procedure are described in Section 6.5.3 of Saad [14].

The optimality property, Equation (3.14), assures that GMRES will return the exact solution when Arnoldi breaks down so GMRES, like FOM, has lucky breakdown. The optimality makes it also possible to prove all kinds of convergence theorems, such as Theorem 2 in Section 3.3. In many cases, convergence is even super linear. The downside of the algorithm is that it depends on all previously calculated basis vectors, which implies that the required memory and the amount of work per iteration increase with each iteration. This problem applies to all Krylov methods that are based on Arnoldi's method, since the basis vectors are needed in the orthonormalization procedure, Equation (3.8).

Consequently, in order to find a Krylov method that has the nice short recurrence property, a different basis for $\mathcal{K}_m$ will have to be formed. Regretly, as was proven by Faber et al. [3], it is impossible to obtain a Krylov method for general matrices that has both the short recurrence and the optimality property. It is therefore known in advance that any method based on this alternative procedure will not produce optimal solutions.

3.2.3 GCR, GMRESR

In practice, the memory problem makes it often necessary to restart the GMRES method after every m iterations, denoted GMRES(m), or to truncate the set of search vectors so that only the most recent n vectors are kept. Standard GMRES is not suitable for truncation, so it will need to be modified. Truncation is generally less detrimental than restarting, but either way, convergence will slow down considerably.

One method that is suitable for truncation is the *Generalized Conjugate Residual* (GCR) method. This method is mathematically equivalent to GMRES, but it constructs an approximate solution $\boldsymbol{x}_m$ during the process instead of afterward. Old basis vectors are therefore used only for constructing new search vectors, and may be dropped after n iterations at the cost of losing optimality. However, since $\boldsymbol{x}_m$ and $\boldsymbol{r}_m$ are constructed simultaneously, GCR requires a double set of vectors in memory. Therefore, real gains are obtained only if the number of iterations is large. Still, the GCR method is in many respects more flexible than GMRES, and there are situations in which GCR is the better choice for reasons other than memory.

The GMRESR method has been proposed in an attempt to combine the strong points of GCR and GMRES. The method is identical to GCR, except that the search vectors are constructed from a second GMRES(m) loop. The idea is that m consecutive basis vectors can be condensed into a single search vector without losing too much information. Since the outer GCR loop uses only these condensed vectors, the memory requirements will be considerably reduced compared with GCR. Moreover, if it is still necessary to truncate GMRESR, the effect will be much smaller.

3.2.4 Biconjugate Gradient Method

When the Arnoldi algorithm is applied to symmetric matrices, matrix $\boldsymbol{H}_m$ becomes tridiagonal. This follows directly from $\boldsymbol{H}_m = \boldsymbol{V}_m^T \boldsymbol{A} \boldsymbol{V}_m$, which is symmetrical if $\boldsymbol{A}$ is, and since $\boldsymbol{H}_m$ is also Hessenberg by construction it must be tridiagonal. Consequently, for symmetric matrices, equation (3.8) simplifies to:

$$\boldsymbol{A}\boldsymbol{v}_m = h_{m-1,m}\boldsymbol{v}_{m-1} + h_{m,m}\boldsymbol{v}_m + h_{m+1,m}\boldsymbol{v}_{m+1}, \quad \boldsymbol{v}_{m-1} \perp \boldsymbol{v}_m \perp \boldsymbol{v}_{m+1}. \tag{3.19}$$

This special case of Arnoldi's method is known as the *Lanczos algorithm*. The above three-term recurrence relation guarantees — at least in exact arithmetic — orthonormality of all basis vectors. Apparently, for symmetric matrices it is possible to have both optimality and short recurrences. Indeed, when $\boldsymbol{A}$ is symmetric positive definite, imposing the FOM condition $\boldsymbol{r}_m \perp \mathcal{K}_m$ leads to the *Conjugate Gradient Method* (CG), which has both these properties.

For general, non-symmetric matrices, the above three-term recurrence can not be used. There exists, however, a similar algorithm for general matrices, known as the *Bi-Lanczos Algorithm*. This algorithm constructs a pair of biorthonormal bases for the subspaces $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{v}_1)$ and $\mathcal{K}_m(\boldsymbol{A}^T, \boldsymbol{w}_1)$. These spaces are identical if $\boldsymbol{A}$ is symmetrical and $\boldsymbol{v}_1 = \boldsymbol{w}_1$, in which case a single, orthonormal basis is formed. Therefore, the Bi-Lanczos algorithm is a generalization of the Lanczos algorithm.

The Bi-Lanczos algorithm is described by the following, implicit recursion:

$$\begin{aligned} \boldsymbol{A}\boldsymbol{v}_m &= \beta_m \boldsymbol{v}_{m-1} + \alpha_m \boldsymbol{v}_m + \delta_{m+1}\boldsymbol{v}_{m+1} \\ \boldsymbol{A}^T\boldsymbol{w}_m &= \delta_m \boldsymbol{w}_{m-1} + \alpha_m \boldsymbol{w}_m + \beta_{m+1}\boldsymbol{w}_{m+1} \end{aligned} \qquad \boldsymbol{v}_i^T \boldsymbol{w}_j = \delta_{ij}, \tag{3.20}$$

from which $\boldsymbol{v}_{m+1}$ and $\boldsymbol{w}_{m+1}$ can be solved. It follows that $\alpha_m = \boldsymbol{w}_m^T \boldsymbol{A} \boldsymbol{v}_m$. Coefficients β_m and δ_m are known from the previous iteration. The new β_{m+1} and δ_{m+1} must be chosen such that $\boldsymbol{v}_{m+1}^T \boldsymbol{w}_{m+1} = 1$, which is a single condition for two unknowns so there are numerous ways to choose them.

A problem occurs when $\boldsymbol{v}_{m+1}$ and $\boldsymbol{w}_{m+1}$ are (nearly) perpendicular, in which case the normality condition can not be met and the method breaks down. If either one of the vectors is zero, the exact solution is part of the Krylov subspace so this is 'lucky breakdown', as found before with FOM and GMRES. However, when the method breaks down due to a zero inner product while both vectors

are non zero, a solution has not yet been found. This situation is called 'serious breakdown' and is of major concern for all Krylov methods that are based on this algorithm.

When the two generated bases are denoted as $\boldsymbol{V}_m$ and $\boldsymbol{W}_m$, the first m Bi-Lanczos iterations can be written as:

$$\begin{aligned} \boldsymbol{A}\boldsymbol{V}_m &= \boldsymbol{V}_m\boldsymbol{T}_m + \delta_{m+1}\boldsymbol{v}_{m+1}\boldsymbol{e}_m^T \\ \boldsymbol{A}^T\boldsymbol{W}_m &= \boldsymbol{W}_m\boldsymbol{T}_m^T + \beta_{m+1}\boldsymbol{w}_{m+1}\boldsymbol{e}_m^T \end{aligned} \qquad \boldsymbol{V}_m^T\boldsymbol{W}_m = \boldsymbol{I}, \tag{3.21}$$

where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{T}_m$ an $m \times m$ tridiagonal matrix which diagonals are formed by the coefficients α, β and δ:

$$\boldsymbol{T}_m = \begin{pmatrix} \alpha_1 & \beta_2 & & \\ \delta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_m \\ & & \delta_m & \alpha_m \end{pmatrix}. \tag{3.22}$$

The *Biconjugate Gradient Method* (BCG) is based on this algorithm. Like the Conjugate Gradient Method, this method is mathematically equivalent to FOM: the residuals are made perpendicular to the Krylov subspaces. Since the Bi-Lanczos algorithm is a generalization of the Lanczos algorithm, this means that the BCG method is a generalization of the CG method.

Using the bi-orthonormality of the generated bases $\boldsymbol{V}_m$ and $\boldsymbol{W}_m$, the same result as Equation (3.12) is found that was derived earlier for FOM. The Hessenberg matrix $\boldsymbol{H}_m$ is replaced by the tridiagonal matrix $\boldsymbol{T}_m$. Substitution into Equation (3.6) yields the following expression for the BCG iterates:

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + \boldsymbol{V}_m\boldsymbol{T}_m^{-1}\beta\boldsymbol{e}_1. \tag{3.23}$$

When the tridiagonal matrix $\boldsymbol{T}_m$ is decomposed in an upper diagonal matrix $\boldsymbol{U}_m$ and a lower diagonal matrix $\boldsymbol{L}_m$, such that $\boldsymbol{T}_m = \boldsymbol{L}_m\boldsymbol{U}_m$, the iterates can be written as $\boldsymbol{x}_m = \boldsymbol{x}_0 + \boldsymbol{V}_m\boldsymbol{U}_m^{-1}\boldsymbol{z}_m$, with $\boldsymbol{L}_m\boldsymbol{z}_m = \beta\boldsymbol{e}_1$. Because of the structure of $\boldsymbol{L}_m$, successive vectors $\boldsymbol{z}_m$ differ only in the last element: $\boldsymbol{z}_m = (\boldsymbol{z}_{m-1}^T, \zeta_m)^T$. This makes it possible to iteratively update the solution and the residual — an obvious necessity for an algorithm that targets short recurrences. Defining a new matrix $\boldsymbol{P}_m = \boldsymbol{V}_m\boldsymbol{U}_m^{-1}$ with columns $\boldsymbol{p}_i$, the iterates $\boldsymbol{x}_m$ and residuals $\boldsymbol{r}_m$ can be written as:

$$\boldsymbol{x}_m = \boldsymbol{x}_{m-1} + \zeta_m\boldsymbol{p}_m \tag{3.24}$$

$$\boldsymbol{r}_m = \boldsymbol{r}_{m-1} - \zeta_m\boldsymbol{A}\boldsymbol{p}_m. \tag{3.25}$$

In order to exploit the biorthonormality of the basis vectors produced by the Bi-Lanczos algorithm, the BCG method simultaneously solves a dual system $\boldsymbol{A}^T\boldsymbol{x}^* = \boldsymbol{b}^*$, with residuals $\boldsymbol{r}_m^* \perp \mathcal{K}_m(\boldsymbol{A}^T, \boldsymbol{w}_1)$. Defining a new matrix $\boldsymbol{P}_m^* = \boldsymbol{W}_m\boldsymbol{L}_m^{-T}$ with columns $\boldsymbol{p}_i^*$, these residuals can again be updated iteratively:

$$\boldsymbol{r}_m^* = \boldsymbol{r}_{m-1}^* - \zeta_m^*\boldsymbol{A}^T\boldsymbol{p}_m^*. \tag{3.26}$$

According to Equation (3.13), the residuals are multiples of the basis vectors $\boldsymbol{v}_m$ and $\boldsymbol{w}_m$, which are biorthonormal. Since $\boldsymbol{V}_m = \boldsymbol{P}_m\boldsymbol{U}_m$ and $\boldsymbol{W}_m = \boldsymbol{P}_m^*\boldsymbol{L}_m^T$,

the residuals can be expressed in terms of search vectors $\boldsymbol{p}_m$ and $\boldsymbol{p}^*_m$, which are $\boldsymbol{A}$-conjugate: $(\boldsymbol{P}^*_m)^T\boldsymbol{A}\boldsymbol{P}_m = \boldsymbol{I}$. This follows directly from the Bi-Lanczos relation $\boldsymbol{W}^T_m\boldsymbol{A}\boldsymbol{V}_m = \boldsymbol{T}_m$. Combined, the following recursive expressions can be obtained:

$$\begin{aligned} \tilde{\boldsymbol{p}}_{m+1} &= \boldsymbol{r}_m + \tilde{\beta}_m\tilde{\boldsymbol{p}}_m \\ \tilde{\boldsymbol{p}}^*_{m+1} &= \boldsymbol{r}^*_m + \tilde{\beta}_m\tilde{\boldsymbol{p}}^*_m \end{aligned} \qquad \tilde{\beta}_m = \frac{(\boldsymbol{r}^*_m, \boldsymbol{r}_m)}{(\boldsymbol{r}^*_{m-1}, \boldsymbol{r}_{m-1})}, \tag{3.27}$$

where $\tilde{\boldsymbol{p}}_m$ and $\tilde{\boldsymbol{p}}^*_m$ are multiples of $\boldsymbol{p}_m$ and $\boldsymbol{p}^*_m$, respectively:

$$\begin{aligned} \zeta_m\boldsymbol{p}_m &= \tilde{\alpha}_m\tilde{\boldsymbol{p}}_m \\ \zeta^*_m\boldsymbol{p}^*_m &= \tilde{\alpha}_m\tilde{\boldsymbol{p}}^*_m \end{aligned} \qquad \tilde{\alpha}_m = \frac{(\boldsymbol{r}^*_{m-1}, \boldsymbol{r}_{m-1})}{(\tilde{\boldsymbol{p}}^*_m, \boldsymbol{A}\tilde{\boldsymbol{p}}_m)}. \tag{3.28}$$

Note in Equations (3.24)–(3.26) that the search vectors $\boldsymbol{p}_m$ and $\boldsymbol{p}^*_m$ are used only in combination with ζ_m and ζ^*_m. Therefore it suffices to form the modified search vectors via Equation (3.27) and calculate the residuals from these. The factors $\tilde{\alpha}_m$ and $\tilde{\beta}_m$ are used for both the original and the dual system, so the solution of this dual system does not double the cost of an iteration.

The resulting Bi-Conjugate Gradient algorithm iteratively updates $\boldsymbol{x}_m$, $\boldsymbol{r}_m$ and $\boldsymbol{r}^*_m$, using the modified search vectors formed from Equation (3.27). Only the most recent search vectors need to be stored, and the two most recent residuals for the calculation of β_m. As was noted before, this short recurrence property can not be united with optimality, except when $\boldsymbol{A}$ is symmetrical, in which case BCG is equivalent to CG. An important problem is the possibility of serious breakdown, inherited from the underlying Bi-Lanczos method. Some look-ahead strategies are developed that allow the algorithm to continue in most cases.

3.2.5 CGS, BICGSTAB

The BCG method simultaneously solves two systems: $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{A}^T\boldsymbol{x}^* = \boldsymbol{b}^*$. The extra calculations involved in solving this dual system are wasted if the dual solution is not used, which is generally the case. Moreover, these calculations involve a multiplication with $\boldsymbol{A}^T$, which can pose problems in situations where $\boldsymbol{A}$ is not explicitly available. For these reasons, variants of the BCG method are developed that do not require this inverse multiplication. These methods use a different procedure to create the basis $\boldsymbol{V}_m$, one that does not depend on a dual basis $\boldsymbol{W}_m$. This leads to an often faster convergence compared to the original BCG method.

One such method if the *Conjugate Gradient Squared* (CGS) method, in which $\boldsymbol{r}_m$ and $\boldsymbol{r}^*_m$ are polynomials of $\boldsymbol{A}$ and $\boldsymbol{A}^T$, respectively. The resulting method does not contain the transposed matrix, and often converges about twice as fast as the BCG method, while using about the same amount of work per iteration. A problem is the highly irregular nature of its convergence, which can lead to a substantial build-up of rounding errors. The *Biconjugate Gradient Stabilized* method (BICGSTAB) is a variation on CGS that was developed to remedy this problem by stabilizing the convergence of the original algorithm.

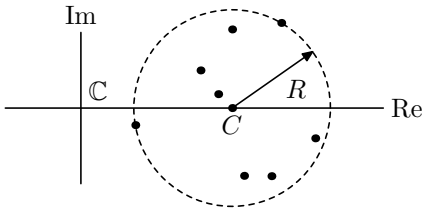

Figure 3.2: Smallest circle containing the spectrum of a matrix $\boldsymbol{A}$. According to Theorem 2, the fraction R/C forms an upper limit for the convergence of the GMRES method.

3.3 Preconditioning

The convergence rate of Krylov subspace methods depends largely on the eigenvalue distribution of the coefficient matrix. In general it can be said that matrices with tightly clustered eigenvalues experience the best convergence. For GMRES, due to its optimality property, Equation (3.14), this notion can be mathematically supported with the following theorem from Saad [14]:

Theorem 2. *Let $\boldsymbol{A}$ be a diagonizable matrix such that $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^{-1}$, where $\boldsymbol{\Lambda}$ is a diagonal eigenvalue matrix. When $K_2(\boldsymbol{X})$ denotes the condition number of $\boldsymbol{X}$, $\mathbb{P}_m$ the space of polynomials of degree less then m and $\sigma(\boldsymbol{A})$ the spectrum of $\boldsymbol{A}$, then the residual norm at the m-th GMRES step satisfies the inequality*

$$\|\boldsymbol{r}_m\|_2 \leq K_2(\boldsymbol{X})\epsilon^{(m)}\|\boldsymbol{r}_0\|_2, \tag{3.29}$$

where

$$\epsilon^{(m)} = \min_{\substack{p\in\mathbb{P}_m \\ p(0)=1}} \max_{\lambda\in\sigma(\boldsymbol{A})} |p(\lambda)|. \tag{3.30}$$

If, moreover, the eigenvalues are enclosed in a circle with center $C \in \mathbb{R}$ and radius $R < C$, like in Figure 3.2, then the following upper bound holds:

$$\epsilon^{(m)} \leq \left(\frac{R}{C}\right)^m. \tag{3.31}$$

Proof: recall from Section 3.2, Equation (3.4), that the m-th Krylov iterate is defined as $\boldsymbol{x}_0 + p_m(\boldsymbol{A})\boldsymbol{r}_0$, where p_m is a polynomial of maximum order $m-1$. Using the diagonalization $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^{-1}$, the iterates can be written as

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + \boldsymbol{X}p_m(\boldsymbol{\Lambda})\boldsymbol{X}^{-1}\boldsymbol{r}_0, \tag{3.32}$$

and their residuals as

$$\boldsymbol{r}_m = \boldsymbol{X}(\boldsymbol{I} - \boldsymbol{\Lambda}p_m(\boldsymbol{\Lambda}))\boldsymbol{X}^{-1}\boldsymbol{r}_0. \tag{3.33}$$

The polynomial p_m depends on the Krylov subspace method used to construct a solution. For GMRES, p_m is the polynomial that minimizes the residual norm:

$$\|\boldsymbol{r}_m\|_2 = \min_{p\in\mathbb{P}_{m-1}} \|\boldsymbol{X}(\boldsymbol{I} - \boldsymbol{\Lambda}p(\Lambda))\boldsymbol{X}^{-1}\boldsymbol{r}_0\|_2 = \min_{\substack{p\in\mathbb{P}_m \\ p(0)=1}} \|\boldsymbol{X}p(\boldsymbol{\Lambda})\boldsymbol{X}^{-1}\boldsymbol{r}_0\|_2. \tag{3.34}$$

Equation (3.29) follows directly from repeated application of the matrix norm property $\|\boldsymbol{M}\boldsymbol{v}\|_2 \leq \|\boldsymbol{M}\|_2\|\boldsymbol{v}\|_2$, with $\epsilon^{(m)} = \min_{p\in\mathbb{P}_m, p(0)=1} \|p(\boldsymbol{\Lambda})\|_2$. Since $p(\boldsymbol{\Lambda})$ is a diagonal matrix, its 2-norm equals the absolute largest diagonal element $|p(\lambda)|$, which gives Equation (3.30). The upper bound (3.31) is a direct application of Zarantonello's lemma, presented in Saad [14] as Lemma 6.4, Section 6.11.2.

□

The theorem shows that the convergence speed depends heavily on the radius of the smallest circle that encloses the eigenvalues of the coefficient matrix. This motivates to transform the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ into a system with the same solution, but with more clustered eigenvalues. This can be done by pre-multiplying the original system with a suitable, non-singular matrix $\boldsymbol{M}^{-1}$:

$$\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{b}. \tag{3.35}$$

This procedure is known as *preconditioning* the system. Krylov subspace methods applied to the new system construct a (near) optimal solution in the Krylov subspace $\mathcal{K}_m(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{M}^{-1}\boldsymbol{r}_0)$ instead of $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0)$. Note that if $\boldsymbol{M}$ equals $\boldsymbol{A}$, the new coefficient matrix $\boldsymbol{M}^{-1}\boldsymbol{A}$ becomes the identity matrix, which eigenvalues are all one. In terms of Theorem 2, $C = 1$ and $R = 0$, which means that GMRES converges to the exact solution in a single iteration.

The full matrix is indeed sometimes used to iteratively refine a solution that was calculated with a direct method. For iterative methods, however, using $\boldsymbol{A}$ as a preconditioner is not feasible, as it is much too expensive to calculate its inverse $\boldsymbol{A}^{-1}$. If instead $\boldsymbol{M}$ is not identical but in a way close to $\boldsymbol{A}$, then the eigenvalues of the new system will still be centered around one. If, moreover, its inverse can be obtained cheaply, then this matrix $\boldsymbol{M}$ will drastically improve the efficiency of the iterative method. In fact, the reliability of iterative techniques seems to depend much more on the quality of the preconditioner, than on the particular choice of Krylov subspace method.

Several methods of constructing a more suitable preconditioning matrix are developed, the most important of which are discussed below.

3.3.1 Basic iterative method

Recall from Section 3.1 that the desired properties for $\boldsymbol{M}$, closeness to $\boldsymbol{A}$ and easy invertability, are the same as found for the iteration matrices of basic iterative methods. From Equation (3.2) it followed that the absolute eigenvalues of $\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}$ should be close to zero, or at least smaller than one. Hence, the eigenvalues of $\boldsymbol{M}^{-1}\boldsymbol{A}$ are concentrated around $C = 1$, within a radius $R < 1$. This makes the iteration matrices of Gauss Jacobi, Gauss Seidel and SOR suitable candidates to be used as preconditioner.

From a different point of view, one can also say that basic iterative methods are accelerated by Krylov subspace methods. This view will prove useful when discussing domain decomposition methods in Section 3.4.

3.3.2 Incomplete decomposition

Another way of defining a preconditioner is to perform a decomposition of the original matrix $\boldsymbol{A}$. The two phases in Gaussian elimination, described in Section 3.1, correspond to solving the lower triangular system $\boldsymbol{Ly} = \boldsymbol{b}$ and the upper triangular system $\boldsymbol{Ux} = \boldsymbol{y}$, where $\boldsymbol{LU} = \boldsymbol{A}$. This decomposition of $\boldsymbol{A}$ in a lower and upper triangular matrix is known as *LU decomposition*. This process is very expensive in terms of work and memory due to fill-in of zero elements. However, it is found that the absolute value of this fill-in often decreases rapidly as the distance to the non-zero elements of $\boldsymbol{A}$ increases. This observation is the main idea behind *Incomplete LU* (ILU) preconditioners.

The ILU decomposition method is identical to LU decomposition, except that in the Gaussian elimination process a fixed subset of zero elements of $\boldsymbol{A}$ is always kept zero. Let these elements be denoted by an index set $\mathcal{P}$. On all positions not in $\mathcal{P}$, $\boldsymbol{A}$ is reconstructed completely from the resulting matrices $\boldsymbol{L}$ and $\boldsymbol{U}$:

$$\boldsymbol{A} = \boldsymbol{LU} + \boldsymbol{R}, \tag{3.36}$$

where $\boldsymbol{L}$ and $\boldsymbol{U}$ are a lower and upper triangular matrix such that $l_{ij} = u_{ij} = 0$ if $(i,j) \in \mathcal{P}$, and $\boldsymbol{R}$ is the residual matrix satisfying $r_{ij} = 0$ if $(i,j) \notin \mathcal{P}$. The idea is that the elements r_{ij} will be small, although this depends largely on the structure of $\boldsymbol{A}$ and the choice of $\mathcal{P}$. One such choice is to define $\mathcal{P}$ as the set of positions where $\boldsymbol{A}$ is zero: $\mathcal{P}_0 = \{(i,j)|a_{ij} = 0\}$. Since this allows absolutely no fill-in elements to be formed, this particular choice of $\mathcal{P}$ is called zero fill-in ILU, or ILU(0).

In some situations, the accuracy of ILU(0) can be insufficient to yield an adequate convergence rate. In these cases it may be necessary to use a subset $\mathcal{P} \subset \mathcal{P}_0$, at the cost of increased work per iteration and denser matrices $\boldsymbol{L}$ and $\boldsymbol{U}$. Ideally, the elements of $\mathcal{P}_0$ that are removed correspond to the largest residual elements r_{ij}, but these are obviously not known in advance. Methods have been developed to predict these locations, based on the sparsity pattern $\mathcal{P}_0$. The resulting preconditioners are denoted as ILU(p), where p is a certain measure of accuracy.

3.4 Domain Decomposition

In practice, the system to solve is often very large. This is certainly the case for the system derived in Chapter 2, which represents a high order finite element discretization in four-dimensional space. This has two major consequences for the solution methods discussed so far. First, memory becomes a problem. Even when sparsity of the coefficient matrix is exploited by using an iterative method, the maximum available memory will limit the size of the simulation. Second, the computation time increases with the size of the system. The best algorithms scale linearly, but there will always be a point at which numerical simulations are no longer feasible due to time constraints.

Both memory and computation time can be reduced by decomposing solution methods into smaller parts that can execute simultaneously on separate computers. The total work load may increase due to this decomposition, but since work is done in parallel the computation time is expected to decrease. Several methods that are suitable for parallelization are developed. The most important can be decomposed into two classes: Schwarz methods and Schur complement methods. Both classes will be discussed in this section. Most of this information is obtained from *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations* by Smith et al. [15].

3.4.1 Schwarz methods

Let Ω denote the total set of computational nodes, i.e. degrees of freedom of the system. This set can be divided into a set of interior nodes $\Omega^\bullet$ and a set of boundary nodes Ω°, such that $\Omega^\bullet \cup \Omega^\circ = \Omega$ and $\Omega^\bullet \cap \Omega^\circ = \varnothing$. Schwarz methods start with decomposing Ω into a number of coherent subdomains Ω_i, such that $\cup_i \Omega_i^\bullet = \Omega^\bullet$. It follows that for two adjacent subdomains $\Omega_i^\bullet \cap \Omega_j \neq \varnothing$, hence the subdomains have a certain amount of overlap.

Several Schwarz methods exist, all based on the same straightforward principle: the unknowns on a subdomain are calculated using only the equations that correspond to its interior nodes. When $\boldsymbol{R}_i$ is a (boolean) restriction matrix that selects the elements corresponding to the interior nodes of subdomain Ω_i, this means that the following subset of equations is selected from the complete system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$:

$$\boldsymbol{R}_i \boldsymbol{A} \boldsymbol{x} = \boldsymbol{R}_i \boldsymbol{b}. \tag{3.37}$$

This system is underdetermined; in order to calculate $\boldsymbol{x}$ in the nodes of $\Omega_i^\bullet$, its solution in $\Omega \backslash \Omega_i^\bullet$ must be already known. This solution is not available. Instead, an approximate solution from a previous iteration is used. This effectively means that the original equation is solved on the smaller domain $\Omega_i^\bullet$, with Dirichlet boundary conditions obtained from previous iterations on surrounding domains.

Let an approximate solution be available as $\boldsymbol{x}_n$. The solution of Equation (3.37) can be written as $\boldsymbol{x} = \boldsymbol{x}_n + \boldsymbol{\delta}_i$, where $\boldsymbol{\delta}_i$ is the vector that must be added to $\boldsymbol{x}_n$ to solve the system. When new values for $\boldsymbol{x}$ are calculated on $\Omega_i^\bullet$ only, the difference vector $\boldsymbol{\delta}_i$ has non-zero values only on these nodes, hence can be written as $\boldsymbol{\delta}_i = \boldsymbol{R}_i^T \boldsymbol{\delta}_{\Omega_i}$. Substitution into Equation (3.37) yields the following system:

$$\boldsymbol{A}_i^\bullet \boldsymbol{\delta}_{\Omega_i} = \boldsymbol{R}_i \boldsymbol{r}_n, \quad \boldsymbol{A}_i^\bullet = \boldsymbol{R}_i \boldsymbol{A} \boldsymbol{R}_i^T, \tag{3.38}$$

where $\boldsymbol{r}_n$ is the residual of the approximate solution $\boldsymbol{x}_n$. Matrix $\boldsymbol{A}_i^\bullet$ is square, and if it is non singular as well this new system can be solved. The resulting local difference $\boldsymbol{\delta}_{\Omega_i}$ can then be transformed back into the global difference vector $\boldsymbol{\delta}_i$ so that it can update the approximate solution $\boldsymbol{x}_n$. Summarizing the entire procedure in a single equation, the global difference vector is calculated as follows:

$$\boldsymbol{\delta}_i = \boldsymbol{B}_i \boldsymbol{r}_n, \quad \boldsymbol{B}_i = \boldsymbol{R}_i^T \boldsymbol{A}_{\boldsymbol{i}}^{\bullet\,-1} \boldsymbol{R}_i. \tag{3.39}$$

This difference vector does not correspond to the global error $\boldsymbol{e} = \boldsymbol{A}^{-1}\boldsymbol{r}_n$, not even on $\Omega_i^\bullet$. Therefore, the updated solution vector will not solve the global system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. However, it can be shown [15] that if $\boldsymbol{A}$ is symmetric and positive definite, $\boldsymbol{\delta}_i$ is a projection of $\boldsymbol{e}$ onto the subspace spanned by the rows of $\boldsymbol{R}_i$, such that the difference is minimal in the $\boldsymbol{A}$ norm given by $\|\boldsymbol{v}\|_{\boldsymbol{A}} = \sqrt{\boldsymbol{v}^T\boldsymbol{A}\boldsymbol{v}}$. This leads to the idea that iteratively correcting the approximate solution with these projected errors will result in a convergent process, and that this process converges to the exact solution $\boldsymbol{x}$.

The iterations can be performed in various ways, resulting in different Schwarz methods. Two important methods are *additive Schwarz* and *multiplicative Schwarz*. Starting with an approximate solution $\boldsymbol{x}_n$ and corresponding residual $\boldsymbol{r}_n$, additive Schwarz simply calculates the errors on each subdomain and uses them to correct the current approximation:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{B}_1\boldsymbol{r}_n + \cdots + \boldsymbol{B}_p\boldsymbol{r}_n. \tag{3.40}$$

Obviously, the order in which $\boldsymbol{x}_n$ is updated does not influence the result, and the various errors can be calculated independently These properties make additive Schwarz very easy to parallelize. Multiplicative Schwarz, on the other hand, bases its calculations on the most recent residual:

$$\begin{aligned} \boldsymbol{x}_{n+\frac{1}{p}} &= \boldsymbol{x}_n + \boldsymbol{B}_1\boldsymbol{r}_n \\ \boldsymbol{x}_{n+\frac{2}{p}} &= \boldsymbol{x}_{n+\frac{1}{p}} + \boldsymbol{B}_2\boldsymbol{r}_{n+\frac{1}{p}} \\ &\vdots \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_{n+\frac{p-1}{p}} + \boldsymbol{B}_p\boldsymbol{r}_{n+\frac{p-1}{p}}. \end{aligned} \tag{3.41}$$

This generally leads to much faster convergence, often about twice as fast, but it lacks the nice properties that additive Schwarz has. Therefore it is much harder to parallelize the multiplicative Schwarz method. An often used solution is to assign each domain a colour, in such a way that two domains with the same colour have no direct influence on each other. This means that domains with the same colour can be processed in parallel. However, it also means that convergence is slowed down, due to the colour-imposed ordering, which opposes the parallel gain. Therefore, an optimal colouring will not contain the fewest possible colours, but will try to balance these two effects.

Figure 3.3 shows the additive and multiplicative Schwarz method applied to the one dimensional Laplace equation. The figure shows two things that appear to be valid in other situations as well. First, as was already mentioned, multiplicative Schwarz converges about twice as fast as additive Schwarz. In this two domain case this relation is even exact, as the multiplicative Schwarz iterates are a subset of the additive Schwarz ones. The second thing the figure shows is the dependency of convergence on overlap. Clearly, a large overlap will lead to faster convergence, whereas no overlap will lead to no convergence at all. Hence the requirement $\cup_i \Omega_i^\bullet = \Omega^\bullet$.

In general, convergence will be poor. Therefore, in practice Schwarz methods are always used in combination with an accelerator. Note that both Schwarz

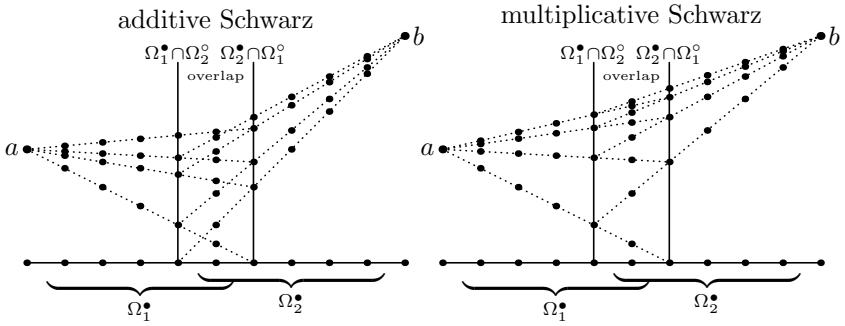

Figure 3.3: Four iterations of additive and multiplicative Schwarz on two subdomains with a zero initial guess, applied to the discretized one dimensional Laplace equation $f''(x) = 0$ with boundary conditions $f(0) = a$ and $f(1) = b$. In this example multiplicative Schwarz converges twice as fast as additive Schwarz. It is clear that for both methods, a large overlap will lead to faster convergence, whereas no overlap will lead to no convergence at all.

methods can be written as:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{B}\boldsymbol{r}_n, \tag{3.42}$$

where

$$\boldsymbol{B} = \begin{cases} \boldsymbol{B}_1 + \cdots + \boldsymbol{B}_p & \text{additive} \\ (\boldsymbol{I} - (\boldsymbol{I} - \boldsymbol{B}_p\boldsymbol{A}) \cdots (\boldsymbol{I} - \boldsymbol{B}_1\boldsymbol{A}))\boldsymbol{A}^{-1} & \text{multiplicative.} \end{cases} \tag{3.43}$$

This explains the origins of the terms 'additive' and 'multiplicative'. With the $\boldsymbol{B}_i$ as defined in Equation (3.39), matrix $\boldsymbol{B}$ turns out to be the inverse of the block-diagonal of $\boldsymbol{A}$ for additive Schwarz, and the inverse of the block lower-triangular matrix for multiplicative Schwarz. Therefore, additive and multiplicative Schwarz can be seen as block variants of the Gauss Jacobi and Gauss Seidel basic iterative methods, respectively.

In practice the matrix $\boldsymbol{B}$ is never composed, since iterative methods need only compute the product $\boldsymbol{y} = \boldsymbol{B}\boldsymbol{x}$, which is performed (in parallel) as Equation (3.40) or (3.41). However, the Schwarz methods do show a large resemblance with the basic iterative methods from Section 3.1. In Section 3.3 it was argued that these methods can be accelerated by a Krylov subspace method by using them as a preconditioner. The same can be done with Schwarz methods, which is the usual way in which they are applied. In this situation, overlap between subdomains is not strictly required.

A problem arises when the subdomain Equations (3.38) are solved using iterative methods, since Krylov subspace methods generally require the preconditioner to be linear and non changing. For non-linear preconditioner, special flexible Krylov subspace methods such as FGMRES must be used. Experience has shown that convergence of the outer iteration is also not affected much when the local problems are solved accurately enough. Another possibility is to use a fixed number of iterations, which makes the operation linear, or to use another

approximate solver such as an incomplete LU decomposition. In that case $\boldsymbol{B}$ becomes a block ILU preconditioner.

3.4.2 Schur complement methods

In the preceding part on Schwarz methods it was noted that the underdetermined Equation (3.37) can only be solved on $\Omega_i^\bullet$ if the solution on $\Omega \setminus \Omega_i^\bullet$ is already known. Actually, it suffices to have a solution on the nodes that are connected to Ω_i, since these are the only values that are manifest in the rows selected by $\boldsymbol{R}_i$. This is evident in Figure 3.3, where the next iteration is determined only by the interface values. Nonetheless, Schwarz methods generate an approximate solution on the entire domain Ω in order to have these values available.

Schur complement methods use the fact that only the interface values are needed during calculations, and solve them separately. The domain Ω is decomposed into a number of subdomains Ω_i, this time with zero overlap, $\Omega_i^\bullet \cap \Omega_j = \varnothing$. This means that also the various submatrices $\boldsymbol{A}_i^\bullet$, as defined in Equation (3.38), have no elements in common. System $\boldsymbol{Ax} = \boldsymbol{b}$ can therefore be written as follows:

$$\begin{pmatrix} \boldsymbol{A}^\circ & \boldsymbol{A}_1^{\circ\bullet} & \dots & \boldsymbol{A}_p^{\circ\bullet} \\ \boldsymbol{A}_1^{\bullet\circ} & \boldsymbol{A}_1^\bullet & & \boldsymbol{0} \\ \vdots & & \ddots & \\ \boldsymbol{A}_p^{\bullet\circ} & \boldsymbol{0} & & \boldsymbol{A}_p^\bullet \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^\circ \\ \boldsymbol{x}_1^\bullet \\ \vdots \\ \boldsymbol{x}_p^\bullet \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}^\circ \\ \boldsymbol{b}_1^\bullet \\ \vdots \\ \boldsymbol{b}_p^\bullet \end{pmatrix}, \tag{3.44}$$

where p is the number of subdomains. From this expression it is clear that the subdomain equations (3.37) can be solved independently when the interface values $\boldsymbol{x}^\circ$ are known:

$$\boldsymbol{x}_i^\bullet = \boldsymbol{A}_i^{\bullet -1}(\boldsymbol{b}_i^\bullet - \boldsymbol{A}_i^{\bullet\circ}\boldsymbol{x}_i^\circ) \tag{3.45}$$

The result is part of the exact solution $\boldsymbol{x}$. Therefore, there is no need to solve the above system more than once, as is the case with Schwarz methods. Instead, Schur methods spend most of their work solving the interface equations, either using a direct or an iterative method. This system of equations is obtained by substituting the above expressions for $\boldsymbol{x}_i^\bullet$ into the full System (3.44):

$$\left(\boldsymbol{A}^\circ - \sum_i \boldsymbol{A}_i^{\circ\bullet}\boldsymbol{A}_i^{\bullet -1}\boldsymbol{A}_i^{\bullet\circ}\right)\boldsymbol{x}^\circ = \boldsymbol{b}^\circ - \sum_i \boldsymbol{A}_i^{\bullet -1}\boldsymbol{b}_i^\bullet, \tag{3.46}$$

or

$$\boldsymbol{S}\boldsymbol{x}^\circ = \boldsymbol{g}. \tag{3.47}$$

Matrix $\boldsymbol{S}$ is known as the *Schur complement matrix*. When the system is solved using an iterative method, the individual matrix elements of $\boldsymbol{S}$ are not required; only multiplications of the form $\boldsymbol{Sx}$ need be performed. Therefore, in that case, the Schur complement matrix is not actually composed. From Equation (3.46) it follows that $\boldsymbol{Sx}$ can be computed as follows:

$$\boldsymbol{Sx} = \boldsymbol{A}^\circ\boldsymbol{x} - \boldsymbol{A}_1^{\circ\bullet}\boldsymbol{v}_1 - \cdots - \boldsymbol{A}_p^{\circ\bullet}\boldsymbol{v}_p, \tag{3.48}$$

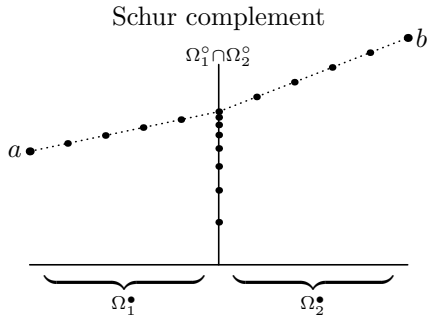

Figure 3.4: The Schur complement method on two subdomains, applied to the same problem as figure 3.3. The subdomains not longer overlap. The interface value is solved using an iterative method, and the remaining internal values are solved after sufficient accuracy is obtained.

where

$$\boldsymbol{A}_i^\bullet \boldsymbol{v}_i = \boldsymbol{A}_i^{\bullet\circ} \boldsymbol{x}. \tag{3.49}$$

These subdomain solves can be computed in parallel. The result is an iterative procedure that computes the interface values $\boldsymbol{x}^\circ$ up to a certain precision. In a second step, the interior values $\boldsymbol{x}_i^\bullet$ are computed via Equation (3.45). This procedure is illustrated in Figure 3.4, which shows the same situation as Figure 3.3.

Chapter 4

Implementation

The Variational Multi-Scale method, introduced in Chapter 2, solves problems of the following kind. Given an initial density, speed and temperature on a three-dimensional domain Ω, simulate the next t_e seconds of the flow on this domain. The eventual goal is to investigate (changing) domains such as the space surrounding an airplane, or a wing, in order to determine its aerodynamic properties. While still under heavy development, however, the current focus is merely non changing, rectangular domains.

Currently, a working implementation of the VMS method already exists. Section 4.1 describes this implementation in detail. Numerical experiments have shown that it performs quite well for small problems that can be solved on a single domain. Larger problems, however, suffer a significant drop in convergence speed. Section 4.2 tries to explain this phenomenon. A possible solution is expected to be found in a deflation method, which is introduced in Section 4.3.

4.1 Current implementation

The current VMS implementation combines most of the numerical methods introduced in Chapter 3. This section describes in detail which methods are used, and — to some extent — why they were chosen over their alternatives. The most important results from previous sections are repeated in order to have a clear overview of the entire procedure. As such, this section can be viewed as a summary of the foregoing material.

The implementation is described in three parts. *Preparation* shows what needs be done once before starting a simulation. *Outer iteration* describes all that must be repeated for each separate time-slab, and finally *Inner iteration* shows the algorithm that is run in each separate Newton iteration.

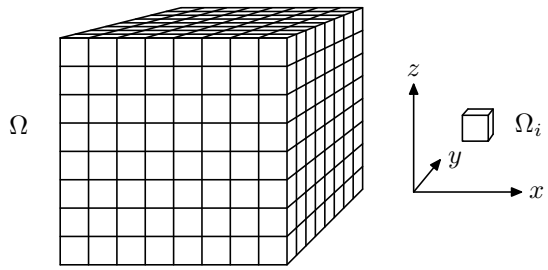

Figure 4.1: A cubic domain Ω, subdivided in $8 \times 8 \times 8$ spatial elements Ω_i. The four-dimensional finite elements Q_i are created by extruding the spatial elements in time by an amount δt.

4.1.1 Preparation

As described in Section 2.4, the current implementation uses the concept of time-slabs, which can be viewed as thin time-slices of the four-dimensional space-time domain $\Omega \times (0, t_e)$ on which the flow is simulated. Figure 2.3 on page 11 gives a schematic overview of this setup. Starting from the initial condition on the spatial domain Ω, the flow during a small time step δt is simulated by solving the equations derived in Chapter 2 on a single time-slab. The state at the end of this simulation is used as an initial condition for the next, and this process is repeated until the required simulation time t_e is reached.

While the VMS method is still under development, it is currently applied only to simple, non changing, rectangular domains, such as shown in figure 4.1. All time-slabs are created by extruding the three-dimensional spatial domain Ω by a fixed amount δt in time: $Q = \Omega \times (0, \delta t)$. Note that Q denotes a single time-slab, not the entire solution space; the suffix n used in Section 2.4 is dropped since all time-slabs are identical. This also means that there will be no difference between the successive VMS calculations, other than initial and boundary conditions.

Since Ω is a rectangular domain, it can easily be decomposed into a number of rectangular elements such that $\Omega = \cup_i \Omega_i$. See again Figure 4.1. The finite elements Q_i are formed by the same straightforward extrusion of Ω_i in time: $Q_i = \Omega_i \times (0, \delta t)$. Recall from Section 2.4 that the solution vector $\boldsymbol{Y} : Q \to \mathbb{R}^5$, containing flow field variables density, speed and temperature, is defined as a sum of locally defined basis functions:

$$\boldsymbol{Y}\big|_{Q_i} = \sum_{a \in \mathcal{A}} N_{(i,a)} \boldsymbol{M}_{(i,a)} \boldsymbol{y}, \tag{4.1}$$

where $\mathcal{A}$ is an index set, $N_{(i,a)} : Q_i \to \mathbb{R}$ are polynomial basis functions defined only on finite element Q_i, $\boldsymbol{M}_{(i,a)}$ are location operators corresponding to element i and basis function a, and $\boldsymbol{y} \in \mathbb{R}^N$ is the solution vector for a single time-slab. The basis functions are derived from a hierarchical modal p-type expansion, Equation (2.21), page 13. Since all time-slabs are identical, these basis functions need to be defined only once.

4.1.2 Outer iteration

Since it makes no difference if the initial state $\boldsymbol{y}_0$ is specified as an initial condition or obtained from a previous iteration, the solution procedure is identical on each time-slab. The solution vector $\boldsymbol{y}$ is calculated using the iterative procedure derived in Section 2.6:

$$\boldsymbol{y}^{k+1} = \boldsymbol{y}^k - \boldsymbol{x}, \tag{4.2}$$

where $\boldsymbol{x}$ solves

$$\left(\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{y}}(\boldsymbol{y}^k, \boldsymbol{y}_0)\right)\boldsymbol{x} = \boldsymbol{G}(\boldsymbol{y}^k, \boldsymbol{y}_0). \tag{4.3}$$

The non-linear function $\boldsymbol{G}$ has been derived in Section 2.5. The latter system is linear, and can be solved using the iterative methods discussed in Chapter 3. This means that individual elements from the Jacobian matrix $\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{y}}$ are not used. Instead, only the matrix-vector multiplications need be performed. It is possible to perform this multiplication without actually composing the Jacobian matrix, as follows:

$$\left(\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{y}}(\boldsymbol{y}^k, \boldsymbol{y}_0)\right)\boldsymbol{v} = \frac{\boldsymbol{G}(\boldsymbol{y}^k + \epsilon\boldsymbol{v}, \boldsymbol{y}_0) - \boldsymbol{G}(\boldsymbol{y}^k, \boldsymbol{y}_0)}{\epsilon}, \tag{4.4}$$

where ϵ is a small constant. This way, each matrix-vector multiplication requires only two evaluations of the non-linear function $\boldsymbol{G}$. However, choosing the right constant ϵ is a problem. If it is chosen too large, the result will be inaccurate because it does not resemble the limit. If it is chosen too small, accuracy is destroyed by rounding errors. Also worth considering is that the transpose matrix is not available, which hinders the use of the BCG method. Variants of BCG, such as BICGSTAB, do not require this transpose matrix, so these methods can be used in combination with the above multiplication.

Since the matrix-vector multiplication needs to be performed multiple times during the iterative procedure, it depends on the complexity of $\boldsymbol{G}$ if it is cheaper to compose the Jacobian matrix explicitly or to use the implicit multiplication. For the problem at hand, numerical experiments have shown that the Jacobian matrix can be re-used for several Newton iterations, Equation (4.2), without much affecting convergence. Often, the matrix can even be re-used on successive time-slabs. This motivates to compose the Jacobian matrix explicitly, using Equation (2.38) derived in Section 2.6, page 17.

Equation (4.3) must be solved within each Newton iteration. As this is a very large system, originating from a high-order discretization on a four-dimensional grid, memory and time constraints force this system to be solved on a parallel computer. Hence, domain decomposition is needed. Looking again at Figure 4.1, it is clear that the rectangular domain can be easily split into several, again rectangular, subdomains, by separating blocks or finite elements Q_i. The two main classes of domain decomposition methods, Schwarz and Schur complement methods, were described in Section 3.4. For Schur methods, the interface equations need to be solved quite accurately, which becomes problematic on high dimension spaces. Schwarz methods are besides easier to implement, and therefore the obvious choice.

Section 3.4 introduced additive and multiplicative Schwarz. Of those, multiplicative Schwarz has the best convergence properties, but is essentially a sequential algorithm. Using a coloring scheme, the equations can be ordered such that many can be evaluated in parallel, but this considerably weakens convergence. The coloring of subdomains also complexifies the total procedure. Additive Schwarz is parallel by nature, and therefore much easier to implement. This is considered to be more important than its weaker convergence, hence the following algorithm is based on additive Schwarz.

4.1.3 Inner iteration

As mentioned in Section 3.4, in practice, Schwarz methods are almost always used in combination with a Krylov subspace method. Section 3.2 identified two different classes of Krylov methods: those with the optimality property and long recurrences (GMRES, GCR), and those with short recurrences but no optimality (BCG, CGS, BICGSTAB). It is known that for general matrices, optimality and short recurrences are mutually exclusive.

Perhaps more important than these two properties, however, is robustness. GMRES enjoys 'lucky breakdown', which means that the algorithm may break down due to zero division, but only after the exact solution is found. The BCG-type methods can break down in other situations as well, so specific measures will need to be taken in order to prevent this from happening. GMRES is more reliable in this respect, and since tests for this particular problem have proved that it converges quite fast on a single domain, the long recurrences do not pose much of a problem. It is favoured over GCR because of its lower memory usage, and because truncation will not be necessary.

Section 3.2 presented the equations for non-preconditioned GMRES. Preconditioned with additive Schwarz, the iterates are constructed from the Krylov subspace $\mathcal{K}_m(\boldsymbol{BA}, \boldsymbol{Br}_0)$ instead of $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0)$, with matrix $\boldsymbol{B}$ as defined in Section 3.4. This changes the algorithm only little. The initial vector is changed into $\boldsymbol{Br}_0$, and multiplication by $\boldsymbol{A}$ is replaced by $\boldsymbol{BA}$. Starting with an initial guess $\boldsymbol{x}_0$, the complete algorithm becomes:

1. $\boldsymbol{w} \leftarrow \boldsymbol{0}$
2. for $\boldsymbol{R}$ in subdomains: do
3. solve $\boldsymbol{RAR}^T\boldsymbol{y} = \boldsymbol{Rb} - \boldsymbol{RAx}_0$ for $\boldsymbol{y}$
4. $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{R}^T\boldsymbol{y}$
5. enddo.
6. $\beta \leftarrow \|\boldsymbol{w}\|_2$
7. $\boldsymbol{v}_1 = \boldsymbol{w}/\beta$
8. for $j = 1, 2, \ldots$: do
9. $\boldsymbol{w} \leftarrow \boldsymbol{0}$
10. for $\boldsymbol{R}$ in subdomains: do
11. solve $\boldsymbol{RAR}^T\boldsymbol{y} = \boldsymbol{RAv}_j$ for $\boldsymbol{y}$
12. $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{R}^T\boldsymbol{y}$

13. enddo.
14. for $i = 1, 2, \ldots, j$: do
15. $h_{i,j} \leftarrow (\boldsymbol{w}, \boldsymbol{v}_i)$
16. $\boldsymbol{w} \leftarrow \boldsymbol{w} - h_{i,j}\boldsymbol{v}_i$
17. enddo.
18. $h_{j+1,j} = \|\boldsymbol{w}\|_2$
19. $\boldsymbol{v}_{j+1} = \boldsymbol{w}/h_{j+1,j}$
20. solve $r_2 = \min_{\boldsymbol{y}\in\mathbb{R}^j} \|\beta\boldsymbol{e}_1 - \bar{\boldsymbol{H}}_j\boldsymbol{y}\|_2$
21. if $r_2 < \epsilon$:
22. break
23. enddo.
24. $\boldsymbol{x} \leftarrow \boldsymbol{x}_0 + \boldsymbol{V}_j\boldsymbol{y}$

This algorithm will now be discussed in detail. It starts with calculating vector $\boldsymbol{w} = \boldsymbol{B}\boldsymbol{r}_0$, lines 1–5. Recall from Section 3.4 that matrix $\boldsymbol{B}$ represents a number of subdomain solves:

$$\boldsymbol{B} = \sum_i \boldsymbol{R}_i^T(\boldsymbol{R}_i\boldsymbol{A}\boldsymbol{R}_i^T)^{-1}\boldsymbol{R}_i \tag{4.5}$$

Lines 3–4 calculate a single term in this addition. Since all terms can be calculated independently of each other, they can be computed in parallel if all subdomains are hosted on separate processors. The subdomains are defined entirely by their restriction matrix $\boldsymbol{R}$. Note in line 3 that matrix $\boldsymbol{A}$ is always preceded by this restriction matrix, hence each processor needs only store the element matrices that are relevant to its own subdomain. This is one of the reasons for using a parallel computer, to reduce the memory load on individual processors.

Line 7 defines the first search vector to be the just calculated vector $\boldsymbol{w}$, normalized. Each iteration that follows (lines 8–23) produces a new search vector by multiplying the most recent search vector $\boldsymbol{v}_j$ by $\boldsymbol{B}\boldsymbol{A}$ (lines 9–13) and taking the orthonormal component to all previously calculated search vectors (lines 14–19). The latter procedure is Arnoldi's method, defined implicitly in Section 3.2, Equation (3.8). The coefficients $h_{i,j}$ are stored in a matrix; they are needed later on in the construction of the solution $\boldsymbol{x}$.

The subdomain equations, line 3 and 11, are not solved exactly. Solving these systems exactly is a lot of work, even though they are much smaller than the full system, and the solution is merely a search vector for the global GMRES algorithm. Using a linear, inexact solver does not affect the GMRES convergence much, and can be much faster. Incomplete LU decomposition was introduced in Section 3.3 as a preconditioner, but since $\boldsymbol{L}\boldsymbol{U}$ closely resembles matrix $\boldsymbol{A}$, it can be used as an inexact solver as well. The resulting solution procedure is non iterative, yet it does not use nearly as much memory as direct methods usually do since ILU is defined to have no or little fill-in.

The above process creates only a basis for the Krylov subspace $\mathcal{K}_j(\boldsymbol{B}\boldsymbol{A}, \boldsymbol{B}\boldsymbol{r}_0)$; it does not yet construct a solution in this subspace. For that, the linear least squares problem at line 20 must be solved. Therefore, usually the system is transformed to upper triangular form by a series of Givens rotations. This

procedure is described in detail in Section 6.5.3 of Saad [14]. The residual norm r_2 is obtained as a by-product of this transformation, which must be performed every iteration to maintain triangularity of the system. Therefore, this residual is available at any time during GMRES iterations.

Line 21 monitors the accuracy of the current-best element of the constructed Krylov subspace. If this accuracy is satisfactory, the loop is broken and the solution is constructed. First $\boldsymbol{y}$ is solved from the least squares problem at line 20, which is a triangular system due to the Givens rotations applied throughout the GMRES iterations. When this vector is obtained, a straightforward summation of Krylov basis vectors yields the solution at line 24.

4.2 Current problems

Consider the situation of a single subdomain 'decomposition'. In this situation, the algorithm defined in Section 4.1 simplifies to ILU-preconditioned GMRES. This becomes clear when all occurrences of the restriction matrix $\boldsymbol{R}$ are removed, which is allowed since it equals identity in this situation. Numerical experiments show that for problems small enough to be handled like this, the algorithm converges very fast. An accurate solution is attained long before GMRES's long recurrences start posing difficulties.

Problems arise, however, when larger problems are examined that require a decomposition in multiple subdomains. The algorithm manages to rapidly smooth out the error of the approximation over each separate subdomain, but as the number of subdomains increases, it seems to be getting more and more trouble to reduce the absolute error. Propagating error information between subdomains seems to be very hard, resulting is such slow convergence that it makes the current algorithm unsuitable for solving problems of realistic scale.

The usual solution to this problem is to add a new communication channel in the form of a coarser grid, on which the same problem is solved. Solutions on both grids are mapped through interpolation and restriction. This smaller problem can be solved on fewer subdomains, giving the smoothing operation a more global effect. Consequently, it will be better capable of reducing the error at reasonable speed. The original, fine grid problem must still be solved in order to smooth out high frequency components that can not be represented on the coarse grid. A typical two-level method combines both problems by adding a coarse grid correction step to every GMRES iteration. This should lead to a faster convergent algorithm. Even better results can be obtained with multilevel methods, which use a series of increasingly coarse grids instead of two.

The main problem with two- or multilevel methods is their complexity. Instead of a single grid, two or several grids will have to be defined, each with a new set of basis functions and mappings to other grids. Much would be gained from a method that accelerates convergence without introducing new grids. The deflation method, subject of the next section, is expected to do just this.

4.3 Deflation

In this section, $\boldsymbol{Ax} = \boldsymbol{b}$ denotes a possibly preconditioned system. Section 3.2 showed that Krylov subspace methods construct approximate solutions of the form $\tilde{\boldsymbol{x}} = \boldsymbol{x}_0 + \boldsymbol{k}$, where $\boldsymbol{k}$ is a Krylov subspace element. The approximations produced by GMRES are optimal in the sense that the distance to the exact solution $\boldsymbol{x}$ is minimized in a certain norm. Consequently, the convergence problems experienced in the current implementation are caused by the Krylov subspace, which apparently is not rich enough to approximate the exact solution to high accuracy, except at very high degree. The convergence problem will therefore apply to all other Krylov subspace methods as well.

To accelerate convergence, the Krylov subspace must be changed such that it is better capable of approximating $\boldsymbol{x} - \boldsymbol{x}_0$. One way of doing this is preconditioning the matrix $\boldsymbol{A}$, as discussed in Section 3.3, which leads to a totally different Krylov subspace. However, the current implementation already uses a combination of Schwartz and ILU preconditioning, and an extra preconditioner of the types discussed so far is expected to either destroy the parallel properties brought by Schwarz, or otherwise not to help much.

Another approach is to augment the Krylov subspace with a fixed set of vectors that it seems to lack. For instance, in the domain decomposition setting brought by the Schwarz preconditioner, a set of vectors $\boldsymbol{z}_i$ with constant value on subdomain i and zeros elsewhere will be able to constitute a global, coarse grid solution, which the current implementation seems to have trouble finding. The previous section argued that the Krylov method performs well in reducing local errors, so together with this global solution the algorithm is expected to converge quite fast. Note the close resemblance of this approach with the two-level methods introduced in the previous section, for this particular choice of augmenting vectors. The actual implementation, however, is quite different.

Let an approximate solution $\tilde{\boldsymbol{x}}$ and its residual $\boldsymbol{r}$ be denoted as:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x}_0 + \boldsymbol{Z}\boldsymbol{\mu} + \boldsymbol{k}, \tag{4.6}$$

$$\boldsymbol{r} = \boldsymbol{r}_0 - \boldsymbol{AZ}\boldsymbol{\mu} - \boldsymbol{Ak}, \tag{4.7}$$

where the columns of $\boldsymbol{Z}$ form a set of linearly independent augmenting vectors, better known as *deflation vectors*. The column space of $\boldsymbol{Z}$ is called the *deflation subspace*. The vectors $\boldsymbol{\mu}$ and $\boldsymbol{k}$ define how $\tilde{\boldsymbol{x}}$ is constructed from both the deflation and a Krylov subspace.

Recall from Section 3.2 that the Full Orthogonalization Method (and derivatives, like the Conjugate Gradient method) sets the residual perpendicular to the Krylov subspace. Seeing the deflation subspace as an augmentation of the Krylov subspace, the same condition is imposed on the deflation vectors: $\boldsymbol{Z}^T\boldsymbol{r} = 0$. If $\boldsymbol{Z}^T\boldsymbol{AZ}$ is non singular, this system can be solved for $\boldsymbol{\mu}$:

$$\boldsymbol{\mu} = \boldsymbol{E}^{-1}\boldsymbol{Z}^T(\boldsymbol{r}_0 - \boldsymbol{Ak}), \quad \boldsymbol{E} = \boldsymbol{Z}^T\boldsymbol{AZ}. \tag{4.8}$$

The non-singularity condition is automatically satisfied if $\boldsymbol{A}$ is positive definite, since for such matrix $\boldsymbol{Z}^T\boldsymbol{AZx} = \boldsymbol{0}$ has only the trivial solution. In general,

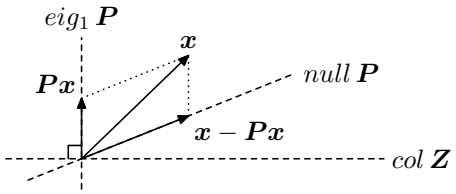

Figure 4.2: Example decomposition of a two-dimensional vector $\boldsymbol{x}$ into a component $\boldsymbol{Px}$ from the eigenspace of $\boldsymbol{P}$ and a component $\boldsymbol{x}-\boldsymbol{Px}$ from the null-space of $\boldsymbol{P}$. The projection operator $\boldsymbol{P}$ is defined in Equation (4.10).

however, $\boldsymbol{E}$ is non singular if and only if

$$(col\,\boldsymbol{Z})^{\perp} \cap (col\,\boldsymbol{AZ}) = \boldsymbol{0}, \tag{4.9}$$

where $(col\,\boldsymbol{Z})^{\perp}$ is the orthogonal complement of the deflation subspace $col\,\boldsymbol{Z}$ in $\mathbb{R}^N$. This is a restriction on the set of possible deflation subspaces. When the above condition is met, substitution of $\boldsymbol{\mu}$ back into Equation (4.7) yields:

$$\boldsymbol{r} = \boldsymbol{P}(\boldsymbol{r}_0 - \boldsymbol{Ak}), \quad \boldsymbol{P} = \boldsymbol{I} - \boldsymbol{AZE}^{-1}\boldsymbol{Z}^T. \tag{4.10}$$

Matrix $\boldsymbol{P}$ is a projection operator: $\boldsymbol{P}^2 = \boldsymbol{P}$, as is easily checked. The following theorem is a general result for projections:

Theorem 3. *Let $\boldsymbol{P} \in \mathbb{R}^{N\times N}$ be a projection operator, i.e. $\boldsymbol{P}^2 = \boldsymbol{P}$. Then:*

$$eig_1\,\boldsymbol{P} \oplus null\,\boldsymbol{P} = \mathbb{R}^N \tag{4.11}$$

Proof: first two intermediate results will be proved via mutual inclusion.

- $eig_1\,\boldsymbol{P} = col\,\boldsymbol{P}$:

  If $\boldsymbol{x} \in eig_1\,\boldsymbol{P}$, then $\boldsymbol{Px} = \boldsymbol{x}$, hence $\boldsymbol{x} \in col\,\boldsymbol{P}$. Reversely, if $x \in col\,\boldsymbol{P}$, then $\boldsymbol{x} = \boldsymbol{Py}$ for some $\boldsymbol{y}$, hence $\boldsymbol{Px} = \boldsymbol{Py} = \boldsymbol{x}$ and $\boldsymbol{x} \in eig_1\,\boldsymbol{P}$.

- $null\,\boldsymbol{P} = col\,(\boldsymbol{I} - \boldsymbol{P})$:

  If $\boldsymbol{x} \in null\,\boldsymbol{P}$, then $\boldsymbol{Px} = \boldsymbol{0}$ hence $(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{x} = \boldsymbol{x}$ and $\boldsymbol{x} \in col\,(\boldsymbol{I} - \boldsymbol{P})$. Reversely, if $\boldsymbol{x} \in col\,(\boldsymbol{I}-\boldsymbol{P})$, then $x = (\boldsymbol{I}-\boldsymbol{P})\boldsymbol{y}$ for some $\boldsymbol{y}$, hence $\boldsymbol{Px} = \boldsymbol{0}$ and $\boldsymbol{x} \in null\,\boldsymbol{P}$.

Equation (4.11) now follows directly from $\boldsymbol{x} = \boldsymbol{Px} + (\boldsymbol{I} - \boldsymbol{P})\boldsymbol{x}$.

□

The theorem states that each projection uniquely decomposes a vector $\boldsymbol{x}$ into an element from its null space and an element from its eigenspace of eigenvalue one. The projection $\boldsymbol{Px}$ returns the latter component; see also Figure 4.2. Consequently, re-projecting the result has no effect. This corresponds to the imposed requirement that $\boldsymbol{P}^2 = \boldsymbol{P}$.

The following theorem is specific for the projection of Equation (4.11):

Theorem 4. *Let $\boldsymbol{A}$ be a non-singular matrix, $\boldsymbol{Z}$ a deflation matrix and $\boldsymbol{P}$ the projection matrix as defined in Equation (4.10). Then:*

1. $eig_1\, \boldsymbol{P} = (col\, \boldsymbol{Z})^{\perp}$
2. $null\, \boldsymbol{P} = col\, \boldsymbol{AZ}$

Proof: the two properties are again proved via mutual inclusion:

- $eig_1\, \boldsymbol{P} = (col\, \boldsymbol{Z})^{\perp}$:

  If $\boldsymbol{x} \in eig_1\, \boldsymbol{P}$, then $\boldsymbol{Px} = \boldsymbol{x}$. From $\boldsymbol{Z}^T\boldsymbol{P} = \boldsymbol{0}$ follows $\boldsymbol{0} = \boldsymbol{Z}^T\boldsymbol{x}$ hence $\boldsymbol{x} \perp \boldsymbol{Z}$. Reversely, if $\boldsymbol{x} \perp \boldsymbol{Z}$, then $\boldsymbol{Z}^T\boldsymbol{x} = \boldsymbol{0}$. From $\boldsymbol{Px} = \boldsymbol{x} - \boldsymbol{AZE}^{-1}\boldsymbol{Z}^T\boldsymbol{x}$ follows $\boldsymbol{Px} = \boldsymbol{x}$, hence $\boldsymbol{x} \in eig_1\, \boldsymbol{P}$.

- $null\, \boldsymbol{P} = col\, \boldsymbol{AZ}$:

  If $\boldsymbol{x} \in null\, \boldsymbol{P}$ then $\boldsymbol{Px} = \boldsymbol{0}$. From $\boldsymbol{Px} = \boldsymbol{x} - \boldsymbol{AZE}^{-1}\boldsymbol{Z}^T\boldsymbol{x}$ follows $\boldsymbol{x} = \boldsymbol{AZE}^{-1}\boldsymbol{Z}^T$ hence $\boldsymbol{x} \in col\, \boldsymbol{AZ}$. Reversely, if $\boldsymbol{x} \in col\, \boldsymbol{AZ}$ then $\boldsymbol{x} = \boldsymbol{AZy}$ for some $\boldsymbol{y}$. From $\boldsymbol{PAZ} = \boldsymbol{0}$ follows $\boldsymbol{Px} = \boldsymbol{0}$, hence $\boldsymbol{x} \in null\, \boldsymbol{P}$.

□

Note that these theorems correspond well with the non-singularity condition Equation (4.9): if $\boldsymbol{P}$ is well defined, then $\boldsymbol{E}$ is non singular. Continuing the normal procedure from Equation (4.10), a Krylov vector $\boldsymbol{k}$ is constructed such that the approximate solution $\tilde{\boldsymbol{x}}$ is close to the exact solution $\boldsymbol{x}$, or equivalently, such that the residual $\boldsymbol{r}$ is close to zero. Equation (4.10) shows that this corresponds to solving the preconditioned, original system:

$$\boldsymbol{PAk} = \boldsymbol{Pr}_0. \tag{4.12}$$

When standard preconditioned Krylov subspace methods are used, the solution $\boldsymbol{k}$ is constructed from $\mathcal{K}(\boldsymbol{PA}, \boldsymbol{Pr}_0)$. Note, however, that the second property of Theorem 4 implies that the preconditioning matrix $\boldsymbol{P}$ is singular. Kaasschieter [6] noted that at least for symmetric and positive definite systems, singular systems can still be solved with the CG method. This solution is not uniquely determined. According to Theorem 4 the null space of $\boldsymbol{P}$ equals the column space of $\boldsymbol{AZ}$, hence $\boldsymbol{r}_0 - \boldsymbol{Ak} \in col\, \boldsymbol{AZ}$, or

$$\boldsymbol{k} = \boldsymbol{A}^{-1}\boldsymbol{r}_0 - \boldsymbol{Z\mu}, \tag{4.13}$$

for some arbitrary vector $\boldsymbol{\mu}$.

To understand the origin of this arbitrary component, recall that Equation (4.6) approximates $\boldsymbol{x} - \boldsymbol{x}_0$ by a sum of this Krylov element $\boldsymbol{k}$ and a deflation subspace element $\boldsymbol{Z\mu}$. If convergence speed was not an issue, the Krylov element would suffice; the deflation vectors are redundant, introduced only to speed up convergence. Both elements are connected through Equation (4.8), but a strict division is not enforced, which explains the arbitrary deflation component in the solution of Equation (4.12).

For a heuristic explanation of why the solution of (4.12) is easier to obtain than that of the original system, consider again the domain decomposition setting with deflation vectors $\boldsymbol{z}_i$ as introduced above: a constant value on subdomain i and zeros elsewhere. Equation (4.13) shows that the solution of the deflated system can contain an arbitrary deflation subspace component. Apparently, subdomain-wide shifts in the solution are ignored, meaning that the solution method focuses only on local errors, which should be much faster. A more mathematical explanation will be given later in this section.

Meanwhile, a solution to the problem is still not available due to the unknown and arbitrary component $\boldsymbol{Z}\boldsymbol{\mu}$. Substitution of $\boldsymbol{k}$ into Equation (4.8) shows that this is in fact the same component as introduced in Equation (4.6), and that it can be extracted by solving the system $\boldsymbol{E}\boldsymbol{\mu} = \boldsymbol{Z}^T(\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{k})$. Matrix $\boldsymbol{E}$ is small and possibly dense, depending on the set of deflation vectors, so a direct method is usually the most suitable choice. When $\boldsymbol{\mu}$ is obtained, substitution into Equation (4.6) yields the solution $\tilde{\boldsymbol{x}}$.

Reviewing the total procedure, the deflation method consists of the following two steps:

1. The original system is solved in such a way that components from the deflation subspace are ignored, leaving the solution with an arbitrary deflation component.
2. This arbitrary component is then replaced with the correct element from the deflation subspace, such that the result approximates $\boldsymbol{x} - \boldsymbol{x}_0$.

The idea is that a well chosen set of deflation vectors will lead to a much faster convergent process. To find out which deflation vectors are most suitable, a number of possible choices will now be examined.

4.3.1 Krylov subspace

An interesting set of deflation vectors is a set that spans the non-preconditioned Krylov subspace of dimension m, such that $col\,\boldsymbol{Z} = \mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0)$. The above procedure suggests that this will not change the total subspace from which a solution is constructed, compared with non-deflated Krylov subspace methods. The following theorem supports this notion.

Theorem 5. *Let $\boldsymbol{A}$ be a non-singular matrix, $\boldsymbol{Z}$ an m-dimensional deflation matrix such that $col\,\boldsymbol{Z} = \mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0)$ for some residual vector $\boldsymbol{r}_0$ and $\boldsymbol{P}$ the projection operator as defined in Equation (4.10). Then, for any $n \in \mathbb{N}$:*

$$\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0) \oplus \mathcal{K}_n(\boldsymbol{P}\boldsymbol{A}, \boldsymbol{P}\boldsymbol{r}_0) \subseteq \mathcal{K}_{m+n}(\boldsymbol{A}, \boldsymbol{r}_0).$$

Proof: since $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0) \subseteq \mathcal{K}_{m+n}(\boldsymbol{A}, \boldsymbol{r}_0)$ and $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0) \perp \mathcal{K}_n(\boldsymbol{P}\boldsymbol{A}, \boldsymbol{P}\boldsymbol{r}_0)$ by assumption, it suffices to proof that $\mathcal{K}_n(\boldsymbol{P}\boldsymbol{A}, \boldsymbol{P}\boldsymbol{r}_0) \subseteq \mathcal{K}_{m+n}(\boldsymbol{A}, \boldsymbol{r}_0)$. This is done by induction.

The Krylov subspace $\mathcal{K}_1(\boldsymbol{PA}, \boldsymbol{Pr}_0)$ can be written as $span\,\{\boldsymbol{Pr}_0\}$. Since $\boldsymbol{Pr}_0 = \boldsymbol{r}_0 - \boldsymbol{AZE}^{-1}\boldsymbol{Z}^T\boldsymbol{r}_0$, with $\boldsymbol{r}_0 \in \mathcal{K}_1(\boldsymbol{A}, \boldsymbol{r}_0)$ and $\boldsymbol{AZE}^{-1}\boldsymbol{Z}^T\boldsymbol{r}_0 \in \boldsymbol{A}\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0) \subseteq \mathcal{K}_{m+1}(\boldsymbol{A}, \boldsymbol{r}_0)$ the inclusion $\mathcal{K}_n(\boldsymbol{PA}, \boldsymbol{Pr}_0) \subseteq \mathcal{K}_{m+n}(\boldsymbol{A}, \boldsymbol{r}_0)$ holds for $n = 1$.

Subspace $\mathcal{K}_{n+1}(\boldsymbol{PA}, \boldsymbol{Pr}_0)$ equals $\boldsymbol{PA}\mathcal{K}_n(\boldsymbol{PA}, \boldsymbol{Pr}_0) \oplus \mathcal{K}_1(\boldsymbol{PA}, \boldsymbol{Pr}_0)$. Assuming that the theorem is true for some n, if $\boldsymbol{v} \in \mathcal{K}_n(\boldsymbol{PA}, \boldsymbol{Pr}_0) \subseteq \mathcal{K}_{m+n}(\boldsymbol{A}, \boldsymbol{r}_0)$, then $\boldsymbol{Av} \in \mathcal{K}_{m+n+1}(\boldsymbol{A}, \boldsymbol{r}_0)$ and $\boldsymbol{AZE}^{-1}\boldsymbol{Z}^T\boldsymbol{v} \in \mathcal{K}_{m+1}(\boldsymbol{A}, \boldsymbol{r}_0)$, hence $\boldsymbol{PAv} \in \mathcal{K}_{m+n+1}(\boldsymbol{A}, \boldsymbol{r}_0)$. This proves the inclusion $\boldsymbol{PA}\mathcal{K}_n(\boldsymbol{PA}, \boldsymbol{Pr}_0) \subseteq \mathcal{K}_{m+n+1}(\boldsymbol{A}, \boldsymbol{r}_0)$. By assumption $\mathcal{K}_1(\boldsymbol{PA}, \boldsymbol{Pr}_0) \subseteq \mathcal{K}_{m+1}(\boldsymbol{A}, \boldsymbol{r}_0)$, which proves $\mathcal{K}_{n+1}(\boldsymbol{PA}, \boldsymbol{Pr}_0) \subseteq \mathcal{K}_{m+n+1}(\boldsymbol{A}, \boldsymbol{r}_0)$.

□

When the Arnoldi process is not broken down, the inclusion becomes an identity because the dimensions on the left and right hand side are equal. Apparently, for this choice of deflation subspace, the constructed solution space is identical to that of a non-deflated Krylov subspace method. Deflation will therefore neither speed up or slow down convergence, the method will simply start at iteration m compared to the non-deflated method. Since it will take m iterations to create the deflation subspace, nothing is gained by this choice of deflation. This was obvious from the start; it was argued before that to speed up convergence, the deflation vectors should not occur in the Krylov subspace until after many iterations. The first m vectors can therefore do no good.

To complete the comparison of deflated and non-deflated Krylov methods, recall that the deflation method was based on the perpendicularity condition $\boldsymbol{r} \perp col\,\boldsymbol{Z} = \mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0)$. FOM and CG use the same condition on $\mathcal{K}_n(\boldsymbol{PA}, \boldsymbol{Pr}_0)$, which by Theorem 5 implies that $\boldsymbol{r} \perp \mathcal{K}_{m+n}(\boldsymbol{A}, \boldsymbol{r}_0)$, hence the obtained solution will be exactly the same for both the deflated and the non-deflated method. GMRES sets $\boldsymbol{r}$ perpendicular to $\boldsymbol{PA}\mathcal{K}_n(\boldsymbol{PA}, \boldsymbol{Pr}_0)$, so for that method the deflated variant will produce a different solution. To be identical in this situation, the deflation condition will have to be changed to $\boldsymbol{r} \perp col\,\boldsymbol{AZ}$, which leads to a new projection operator $\tilde{\boldsymbol{P}}$:

$$\tilde{\boldsymbol{P}} = \boldsymbol{I} - \boldsymbol{F}(\boldsymbol{F}^T\boldsymbol{F})^{-1}\boldsymbol{F}^T, \quad \boldsymbol{F} = \boldsymbol{AZ}. \tag{4.14}$$

4.3.2 Eigenvalue deflation

To get a better insight in the convergence behaviour of deflated Krylov subspace methods, it is useful to focus on the eigenvalues of the deflated matrix $\boldsymbol{PA}$. For a general deflation subspace, the two properties of Theorem 4 are equivalent to:

1. $(\boldsymbol{PA})\boldsymbol{v} = \boldsymbol{Av} \;\Leftrightarrow\; \boldsymbol{Av} \perp col\,\boldsymbol{Z}$
2. $(\boldsymbol{PA})\boldsymbol{v} = \boldsymbol{0} \;\Leftrightarrow\; \boldsymbol{v} \in col\,\boldsymbol{Z}$

Let the eigenvectors of $\boldsymbol{A}$ be denoted $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N$, with corresponding eigenvalues $\lambda_1, \ldots, \lambda_N$. According to the first property, $\boldsymbol{v}_i$ is an eigenvector of the deflated matrix $\boldsymbol{PA}$ as well, with unchanged eigenvalue λ_i, if it is orthogonal to the

deflation subspace $col\,\boldsymbol{Z}$. Hence, when the deflation subspace is defined as the orthogonal complement of the subspace spanned by a set of eigenvectors $\boldsymbol{v}_{i_1}, \ldots, \boldsymbol{v}_{i_{N-m}}$, the eigenvalues of $\boldsymbol{PA}$ are $\lambda_{i_1}, \ldots, \lambda_{i_{N-m}}$ and 0, the latter with multiplicity m due to property two. The eigenvalues that have disappeared are said to be deflated, which explains the name of this process: *eigenvalue deflation*.

To study the convergence properties of the thus deflated algorithm, let a matrix $\boldsymbol{V}$ be formed of a subset of linearly independent eigenvectors, $\boldsymbol{V} = (\boldsymbol{v}_{i_1} \ldots \boldsymbol{v}_{i_N})$. When the deflation subspace is defined as above, $col\,\boldsymbol{Z} = (col\,\boldsymbol{V})^{\perp}$, then this subspace and $col\,\boldsymbol{V}$ are linearly independent. Since $\boldsymbol{A}$ is non singular, the linear independence applies to $col\,\boldsymbol{AZ}$ and $col\,\boldsymbol{AV}$ as well. Spanned by eigenvectors of $\boldsymbol{A}$, the column space of $\boldsymbol{V}$ is an invariant subspace, which implies that $col\,\boldsymbol{AV} = col\,\boldsymbol{V} = (col\,\boldsymbol{Z})^{\perp}$. Equation (4.9) now shows that matrix $\boldsymbol{E}$ in the definition of $\boldsymbol{P}$ is non singular, so the projection operator $\boldsymbol{P}$ is well defined for this particular deflation subspace.

Since the columns of $\boldsymbol{Z}$ and $\boldsymbol{V}$ are all linearly independent eigenvectors of $\boldsymbol{PA}$, the matrix $\boldsymbol{X} = \begin{pmatrix} \boldsymbol{Z} & \boldsymbol{V} \end{pmatrix}$ diagonalizes $\boldsymbol{PA} = \boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^{-1}$, where $\boldsymbol{\Lambda}$ is its diagonal eigenvalue matrix:

$$\boldsymbol{\Lambda} = \begin{pmatrix} \boldsymbol{\Lambda}_Z & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Lambda}_V \end{pmatrix}. \tag{4.15}$$

The submatrices $\boldsymbol{\Lambda}_Z$ and $\boldsymbol{\Lambda}_V$ contain the eigenvalues corresponding to $\boldsymbol{Z}$ and $\boldsymbol{V}$, respectively, such that $\boldsymbol{PAZ} = \boldsymbol{Z}\boldsymbol{\Lambda}_Z$ and $\boldsymbol{PAV} = \boldsymbol{V}\boldsymbol{\Lambda}_V$. Clearly, $\boldsymbol{\Lambda}_Z = \boldsymbol{0}$. With matrices $\boldsymbol{X}$ and $\boldsymbol{\Lambda}$ defined like this, Theorem 2 in Section 3.3 provides an upper bound for the residual at GMRES iteration m. However, since in Equation (3.30) the set of polynomials is restricted to $p(0) = 1$, the zero eigenvalues in $\boldsymbol{\Lambda}_Z$ seem to render this bound useless.

To obtain a useful upper bound for the convergence of deflated GMRES, some of the proof of Theorem 2 needs to be redone, using the fact that the deflated residual $\boldsymbol{Pr}_0$ is an element of $col\,\boldsymbol{P} = col\,\boldsymbol{V}$. Writing $\boldsymbol{Pr}_0$ as $\boldsymbol{Vy}$, Equation (3.34) reduces to:

$$\|\boldsymbol{r}_m\|_2 = \min_{\substack{p \in \mathbb{P}_m \\ p(0)=1}} \left\| \begin{pmatrix} \boldsymbol{Z} & \boldsymbol{V} \end{pmatrix} \begin{pmatrix} p(\boldsymbol{\Lambda}_Z) & \boldsymbol{0} \\ \boldsymbol{0} & p(\boldsymbol{\Lambda}_V) \end{pmatrix} \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{y} \end{pmatrix} \right\|_2 = \min_{\substack{p \in \mathbb{P}_m \\ p(0)=1}} \|\boldsymbol{V} p(\boldsymbol{\Lambda}_V)\boldsymbol{y}\|_2, \tag{4.16}$$

which leads to the upper bound

$$\|\boldsymbol{r}_m\|_2 \leq \epsilon^{(m)} \|\boldsymbol{V}\|_2 \|\boldsymbol{y}\|_2, \tag{4.17}$$

where $\epsilon^{(m)}$ is identical to the constant defined in Theorem 2 except that the eigenvalues corresponding to the columns of $\boldsymbol{Z}$ are ignored. Note that the fact that these eigenvalues are zero is not used. Therefore, more generally it can be said that Theorem 2 can ignore all eigenvalues which corresponding eigenvector is not part of the initial residual. The same holds for the the upper bound for $\epsilon^{(m)}$, Equation (3.31), in which the circle of Figure 3.2 needs only enclose the remaining eigenvalues.

These findings agree with the general explanation of the super linear convergence that is often found in Krylov subspace methods, namely that eigenvectors

that are in a way 'found' during the process do no longer affect convergence, leading to a gradually decreasing *effective condition number* and a correspondingly increasing convergence speed. Eigenvalue deflation can be seen as a means to point the algorithm at a set of eigenvectors that are notoriously hard to find, thus lowering the effective condition number from the start. These eigenvectors typically correspond to the lowest few eigenvalues of the coefficient matrix.

The problem that remains is how to construct a set of deflation vectors such that their orthogonal complement is an invariant subspace of $\boldsymbol{A}$, spanned by the eigenvectors which eigenvalues are not to be deflated. In the special case that $\boldsymbol{A}$ is symmetric this is simplified a bit by the fact that its eigenvectors form an orthogonal basis of $\mathbb{R}^N$, which means that the deflation subspace must itself be an invariant subspace corresponding to the deflated eigenvalues. Though still hard to find, Vuik et al. [18] used a technique called *physical deflation* to construct a basis for this subspace in a diffusion problem with strongly varying coefficients. A generally applicable technique, however, is not available.

The orthogonality of $col\,\boldsymbol{Z}$ and $col\,\boldsymbol{V}$ that follows from symmetry of $\boldsymbol{A}$ does enable a completely different solution method, one that does not require the projection matrix $\boldsymbol{P}$ nor the correction step to remove an arbitrary deflation subspace component from the solution. Note that in Equation (4.6), $\boldsymbol{k}$ is an element of the Krylov subspace $\mathcal{K}(\boldsymbol{PA}, \boldsymbol{Pr}_0)$. A key observation is that with the columns of $\boldsymbol{V}$ being eigenvectors of both $\boldsymbol{A}$ and $\boldsymbol{PA}$, the same holds for all exponentials: $(\boldsymbol{PA})^m\boldsymbol{V} = \boldsymbol{A}^m\boldsymbol{V}$. Since the deflated residual $\boldsymbol{Pr}_0$ is an element of $col\,\boldsymbol{V}$, it follows that

$$\mathcal{K}_m(\boldsymbol{PA}, \boldsymbol{Pr}_0) = \mathcal{K}_m(\boldsymbol{A}, \boldsymbol{Pr}_0). \tag{4.18}$$

When $\boldsymbol{A}$ is symmetric, $\boldsymbol{P}$ becomes an orthogonal projection. Therefore it is equal to the projection used by Arnoldi's method, when it makes a newly calculated vector perpendicular to all previously calculated vectors. Consequently, when Arnoldi's method is modified such that it creates an orthonormal basis for $col\,\boldsymbol{Z} \oplus \mathcal{K}(\boldsymbol{A}, \boldsymbol{r}_0)$, the solution space does not change. When a solution is constructed by FOM, it will indeed be identical to the one found with eigenvalue deflation since then $\boldsymbol{r} \perp col\,\boldsymbol{Z}$. For other Krylov methods the results may not be equal.

The resulting method starts with a set of eigenvectors, constructs an orthonormal basis for the spanned subspace and then switches over to the Krylov subspace $\mathcal{K}(\boldsymbol{A}, \boldsymbol{r}_0)$. Although strictly spoken symmetry is required for Arnoldi's method to project vectors to the span of the remaining eigenvectors, this method seems to yield good results for general matrices as well. Combined with GMRES, this is known as *augmented GMRES*, described a.o. by Morgan [8]. This method uses information about the eigenvectors that is generated during GMRES to construct a set of approximate eigenvectors, corresponding to the smallest eigenvalues, and uses these to augment the Krylov subspace after a restart. This way, the detrimental effects that restarting has on convergence can be reduced considerably.

4.3.3 Subdomain deflation

The physical deflation vectors found by Vuik et al. [18] for a symmetric matrix $\boldsymbol{A}$ constitute a base of continuous, piecewise linear vectors. It was found later [17] that when this set is replaced with a (larger) set that spans the space of piecewise constant vectors, convergence improves, even though this is no longer an invariant subspace of $\boldsymbol{A}$. Apparently this type of deflation still decreases the effective condition number, meaning that whatever non-zero eigenvalues remain are more clustered than the original spectrum.

In a domain decomposition setting, the space of vectors that are constant on subdomains is the column space of the matrix which entries are defined by:

$$z_{ij} = \begin{cases} 1 & i \in \Omega_j^\bullet \\ 0 & i \notin \Omega_j^\bullet, \end{cases} \tag{4.19}$$

where the $\Omega_j^\bullet$ are disjoint index sets such that $\cup_j \Omega_j^\bullet = \Omega^\bullet$. The matrix $\boldsymbol{E}$ in the definition of $\boldsymbol{P}$, Equation (4.10), is defined as $\boldsymbol{Z}^T\boldsymbol{A}\boldsymbol{Z}$. With the above definition of $\boldsymbol{Z}$ this evaluates to

$$e_{mn} = \sum_{(i,j) \in \Omega_m^\bullet \times \Omega_n^\bullet} a_{ij}. \tag{4.20}$$

It follows that if $\boldsymbol{A}$ is a sparse matrix, part of this sparsity will be inherited by the matrix $\boldsymbol{E}$. As a system of the form $\boldsymbol{E}\boldsymbol{v} = \boldsymbol{f}$ must be solved each time that the deflation operator $\boldsymbol{P}$ is applied, the work that is required for deflating a vector decreases considerably due to this sparsity.

Note that the above defined deflation subspace was introduced earlier in this section to clarify some aspects of the deflation method. It was argued that an element $\boldsymbol{Z}\boldsymbol{\mu}$ from this subspace represents a global, coarse grid solution, which means that global errors of this form are ignored by the deflated method. Since global errors are often the bottleneck for block-preconditioned methods, this type of deflation, called *subdomain deflation*, is expected to speed up convergence considerably. Mathematical proofs such as the one obtained for eigenvalue deflation, however, are much harder to find for this type of deflation.

For symmetric matrices, some proofs do exist. Nicolaides [11] showed that the effective condition number is at least as small as the condition number of the original matrix, for a general set of deflation vectors. A stricter bound, specific for subdomain deflation, was provided by Frank and Vuik [4], under the extra condition that $\boldsymbol{A}$ is an M-matrix. They inferred that "subdomain deflation effectively decouples the original system into a set of independent Neumann problems on the subdomains, with convergence governed by the 'worst-conditioned' Neumann problem". This corresponds nicely with the view that subdomain-wide errors are ignored, and that this causes the deflated method to converge faster.

The fact that $\boldsymbol{Z}\boldsymbol{\mu}$ can be viewed as a coarse-grid solution makes subdomain deflation very similar to the two-level methods described in the previous section. Nabben and Vuik [10] proved, again only for the symmetric case, that when the same matrix $\boldsymbol{Z}$ is used for both the restriction and interpolation operation, the

effective condition number of the deflated matrix is always less or equal than the one with coarse grid correction. This means that the deflated method can always converge faster, at the cost of increased work per iteration due to the deflation operation. Since deflation is generally also the simplest of the two methods in terms of implementation, it makes sense to focus the search for a solution to the currently experienced VMS problems on deflation methods.

Chapter 5

Future research

The deflation method that was introduced in the previous chapter has been widely tested, and is shown to give good results in many applications. It is, however, hard to predict its effectiveness when applied to problems for which no numerical results are available. This is certainly the case for the non-symmetric systems that originate from the VMS method described in Chapter 2, because little is known about the general case. Most theorems apply only to symmetric matrices, and also most numerical tests are performed on symmetric matrices only. It will therefore be interesting to see how deflation performs when it is applied to the VMS method. This chapter lists some points of research.

The deflation method is supposed to solve the convergence problem that arises from Schwarz-preconditioning the system. This is a domain decomposition method that makes it possible to perform multiple computations simultaneously on separate computers, which should eventually lower computing times. Also memory requirements will be lower as different parts of the solution can be hosted on different computers. For problems small enough to fit on a single computer, however, the subdomain-deflated algorithm can well be tested sequentially, i.e. on a single computer. This way the pure performance of the various algorithms can be measured, not accounting for differences in parallelizability. These can be examined further after the most promising algorithms have been identified.

A distinctive element of deflation algorithms is the construction of a deflation subspace. The previous chapter introduced subdomain deflation as the deflation-counterpart of course-grid correction. This method has the advantage over eigenvalue deflation that it does not require a set of eigenvectors, and that the sparsity pattern of $\boldsymbol{A}$ is transferred over to $\boldsymbol{E}$. Moreover, tests have shown that subdomain deflation performs often as good or even better than eigenvalue deflation. Research will therefore focus mainly on this type.

In addition to deflation vectors that are constant over a subdomain, Verkaik [16] proposed in his master's thesis to use linearly changing deflation vectors. This

richer deflation subspace will be better able to approximate eigenvectors corresponding to small eigenvalues, and is therefore expected to result in a faster convergent method. This idea can be extended even further by adding second and higher order deflation vectors. The basis functions used in this implementation are of high order as well, and the two may go well together. The system that must be solved for each application of projection $\boldsymbol{P}$ will, however, increase drastically in size. The question is therefore which deflation order will yield the best balance in convergence speed and work per iteration.

Finally there is the question how exactly to apply the deflation technique, because deflation is not a strict mathematical concept. The method that is most often used in publications is based on the projection defined in Equation (4.10). The matrix $\boldsymbol{P}$ is used to precondition the system, and the deflation subspace component is calculated separately. This method, however, is tested mostly on symmetric matrices and hence in combination with the Conjugate Gradient method. The previous chapter showed that the projection matrix looks a little different when the GMRES method is used in its derivation. Since the current implementation is based on GMRES, it will be interesting to examine this projector as well.

Another often used method is augmented deflation, in which no projection is used but the deflation vectors are added directly to the Arnoldi process. This method is normally used in combination with GMRES, but not in in a subdomain setting. Instead, approximate eigenvectors are are added after a restart to speed up convergence, so this method is more like eigenvalue deflation. It will, however, be interesting to see what the effect is of augmented deflation based on the above described subdomain deflation vectors.

Instead of trying these deflation methods directly in the VMS method, which is reasonably complicated, useful experience can be gained from a simpler and better known system. The two-dimensional Laplace equation, discretized using finite differences on a square grid is an obvious choice. In order to stay close to the eventual implementation, the symmetry of the coefficient matrix will be ignored and the system will be solved using GMRES and a block ILU preconditioner. This should provide some experience in the candidate deflation methods, as well as the programming framework in which they will be implemented.

Next, deflation will be applied to the VMS method. A working VMS implementation is available and will be used as a testing ground for the various deflation methods. This implementation is well tested and therefore a trusted data source against which not only convergence rates can be compared, but also consistency of numerical results can be checked. A consequence is that the test problems will be limited to non-changing, rectangular domains such as shown in Figure 4.1, page 37. From this domain, the upper and lower surface will represent a wall with no-slip condition; the other four are periodic, so a flow between two infinite planes is simulated. Near the walls a higher resolution will be required because of the small vortices in that area, and the domain must be large enough to ensure that the turbulence is sufficiently decorrelated in all directions. The resulting flow problem is known as planar channel flow, which is a widely used test case for the research of turbulent flow.

Bibliography

[1] John D. Anderson, Jr. *Fundamentals of Aerodynamics*. McGrawl-Hill, New York, 2001.

[2] S. Scott Collis. The dg/vms method for unified turbulence simulation. In *32nd AIAA Fluid Dynamics Conference and Exhibit*, June 2002.

[3] V. Faber and T. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM Journal on Numerical Analysis*, 21:356–362, 1984.

[4] J. Frank and C. Vuik. On the construction of deflation-based preconditioners. *SIAM Journal on Scientific Computing*, 23(2):442–462, 2000.

[5] Thomas J.R. Hughes, Luca Mazzei, and Kenneth E. Jansen. Large eddy simulation and the variational multiscale method. *Computing and Visualization in Science*, vol. 3(no. 1/2):47–59, 2000.

[6] E.F. Kaasschieter. Preconditioned conjugate gradients for solving singular systems. *Journal of Computational and Applied Mathematics*, 24:265–275, 1988.

[7] George Em Karniadakis and Spencer J. Sherwin. *Spectral/hp element methods for CFD*. Oxford University Press, Oxford, 1999.

[8] Ronald B. Morgan. A restarted gmres method augmented with eigenvectors. *SIAM Journal on Matrix Analysis and Applications*, 16(4):1154–1171, 1995.

[9] E.A. Munts, S.J. Hulshoff, and R. de Borst. A space-time variational multiscale discretization for les. In *34th AIAA Fluid Dynamics Conference and Exhibit*, June 2004.

[10] R. Nabben and C. Vuik. A comparison of deflation and coarse grid correction applied to porous media flow. *SIAM Journal on Numerical Analysis*, 42(4):1631–1647, 2004.

[11] R. Nicolaides. Deflation of conjugate gradients with applications to boundary value problems. *SIAM Journal of Numerical Analysis*, 24, 1987.

[12] Stephen B. Pope. *Turbulent Flows*. Cambridge University Press, Cambridge, 2000.

[13] Lewis F. Richardson. The supply of energy from and to atmospheric eddies. *Proceeding of the Royal Society*, A97, 1920.

[14] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 2000.

[15] Barry F. Smith, Petter E. Bjørstad, and William Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, Cambridge, 1996.

[16] Jarno Verkaik. Deflated krylov-schwarz domain decomposition for the incompressible navier-stokes equations on a colocated grid. Master's thesis, Delft University of Technology, 2002.

[17] F.J. Vermolen, C. Vuik, and A. Segal. Deflation in preconditioned conjugate gradient methods for finite element problems. 2002.

[18] C. Vuik, A. Segal, and J.A. Meijerink. An efficient preconditioned cg method for the solution of a class of layered problems with extreme contrasts in the coefficients. *Journal of Computational Physics*, 152:385–403, 1999.

Index