Iterative solution methods for the simulation of flow in industrial glass furnaces

Eline Jonkers

Master's Thesis

February 2006

Delft University of Technology
TNO Science and Industry

Thesis Committee:
Prof.dr.ir. P. Wesseling (TU Delft)
Dr.ir. C. Vuik (TU Delft)
Dr. D. Hegen (TNO Science and Industry)
Dr. A. Twerda (TNO Science and Industry)

PREFACE

This Master's thesis is written for the degree of Master of Science in Applied Mathematics, faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology. The graduation work is done in the unit of Numerical Analysis, taking a total of nine months of work.
The Master's project is carried out at TNO Science and Industry, department of Process Modeling and Control, business unit Industrial Modeling and Control.

The first three months of the project focused on problem definition, study of literature and planning the research for the next six months. The work done in this period is reported in [7]. The remaining six months of the graduation work were spent on the actual research. The outcome is this Master's thesis.

I would like to thank all members of the committee for their help during the project. Especially I would like to thank Kees Vuik for his excellent supervision and careful reading of intermediate reports and Aris Twerda for all his help with X-stream. To conclude I would like to thank my colleagues of Process Modeling and Control and Instrument Modeling and Control for making my stay at TNO very pleasant.

Delft, February 2006 Eline Jonkers

CONTENTS

1. INTRODUCTION

Mathematical simulation of flows is important for design, optimization and trouble shooting of glass melting furnaces. To gain insight into the glass melting process, physical experiments can be done. However, such experiments are often very costly and time-consuming, and there are circumstances under which certain physical quantities cannot be measured. Simulation by Computational Fluid Dynamics (CFD) does not have these disadvantages. Although CFD only approximates the real physics, it gives engineers in the glass industry great insight into the transport phenomena occurring in glass melting furnaces.

At TNO Science and Industry, a CFD simulation package called X-stream is developed for the glass industry. X-stream is able to simulate the glass melt and the combustion space simultaneously. Besides the computation of the flow fields and the energy equation, there are for instance models for turbulence, combustion and radiation.

To simulate the complex physical processes in a furnace with a high accuracy, many equations have to be solved, resulting in large computing times. A domain decomposition algorithm can be used to divide the total problem into smaller problems, which can be computed in parallel on different processors. Some of the equations that have to be solved are Navier-Stokes equations, equations that describe the conservation of momentum. Because these equations are nonlinear, solving them is very time-consuming. Therefore, we focus in this research primarily on the Navier-Stokes equations.

The main goal of the Master's project research is to get experience with the algorithms used in X-stream and to improve them. To achieve this, Krylov subspace acceleration combined with deflation is considered. Also some attention is paid to Krylov subspace acceleration combined with the Semi-Implicit Method for Pressure-Linked Equations (SIMPLE).

The structure of this thesis is as follows. In Chapter 2 the mathematical model for flow is discussed briefly, followed by the numerical model. In Chapter 3 a discussion of iterative methods for solving the system of equations arising from discretization is presented, followed by an introduction on domain decomposition methods in Chapter 4. In Chapter 5 the connection between X-stream and the subjects treated in the previous chapters is given. Test cases in X-stream used for experiments with deflation methods are described in Chapter 6. In Chapter 7 numerical experiments done with deflation methods and their results are described. Chapter 8 states the GCR-SIMPLE method. Finally, in Chapter 9 conclusions and recommendations are given.

2. FINITE VOLUME DISCRETIZATION

Incompressible flow, for example in glass furnaces, can be described by a mathematical model. From this model, partial differential equations (PDEs) arise. Some PDEs can not be solved directly. These equations have to be solved by transforming them into a finite number of difference equations and then solving this system of equations.
There are a number of ways to transform a system of PDEs into a finite number of difference equations. In CFD, the Finite Difference, Finite Element and Finite Volume method are most commonly used. In this chapter the Finite Volume (FV) method is discussed. The FV method is treated in great detail in for example Wesseling [32], Patankar [13] and Ferziger & Perić [3].

This chapter begins with describing the mathematical model for incompressible flow in Section 2.1. The discretization of the convection-diffusion equation and the incompressible Navier-Stokes equations are described in Section 2.2 and Section 2.3, respectively. These equations play an important role in this thesis.

2.1 Mathematical model of a flow

In this section, the mathematical model for incompressible flow is described. First, some notation is introduced (this notation is used throughout the whole thesis), then physical conservation equations are discussed briefly.

2.1.1 Notation

In this thesis the notation of Wesseling [32] is adopted: we assume a right-handed Cartesian coordinate system $(x_1, x_2, ..., x_d)$ with d the number of space dimensions. Greek letters denote scalars and boldfaced Latin letters denote vectors, for example $\mathbf{x} = (x_1, x_2, ..., x_d)$. In *Cartesian tensor notation*, differentiation is denoted as follows:

$$\phi_{,\alpha} = \frac{\partial \phi}{\partial x_\alpha}.$$

Greek subscripts refer to coordinate directions and the *summation convention* is used: summation takes place over Greek indices that occur twice in a term or product, for example:

$$u_\alpha v_\alpha = \sum_{\alpha=1}^{d} u_\alpha v_\alpha.$$

Sometimes, we will use vector notation if this is more convenient. For example, in vector notation, the divergence of a vector field is written as div $\mathbf{u}$.

2.1.2 Conservation equations

In this subsection the conservation equations for mass, momentum and energy will be given without derivation. The derivation can be found in for example Wesseling [32]. The conservation equations are conveniently arranged in Twerda [18] and Verweij [21].

Conservation of mass

The mass conservation law or continuity equation expresses the principle that the rate of change of mass in an arbitrary material volume equals the rate of mass production in this volume. Usually, in practice, there is no mass production, and then the mass conservation law reads:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_j)}{\partial x_j} = 0, \tag{2.1}$$

where ρ is the density and u_j is the x_j-component of the velocity u.

Conservation of momentum

Conservation of momentum of the flow in each direction is given by

$$\frac{\partial (\rho u_i)}{\partial t} + \frac{\partial (\rho u_i u_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\mu \frac{\partial u_i}{\partial x_j}\right) + \rho g_i, \quad i = 1, \ldots, d, \tag{2.2}$$

where p is the pressure, μ the viscosity and g_i the x_i-component of the gravitational acceleration vector. These equations are also known as the *Navier-Stokes* equations.

Conservation of energy

Energy per unit mass is contained in the flow as internal energy e and kinetic energy ($\frac{1}{2}{u_i}^2$). According to Verweij [21] and Wesseling [32] it is often convenient to solve the enthalpy equation rather than the energy equation. The enthalpy equation reads:

$$\frac{\partial (\rho h - p)}{\partial t} + \frac{\partial (\rho u_j h)}{\partial x_j} = \frac{\partial}{\partial x_j}\left(\frac{\lambda}{c_p}\frac{\partial h}{\partial x_j}\right) + S_{rad}, \tag{2.3}$$

with $h = e + \frac{p}{\rho}$ the enthalpy, c_p the specific heat, λ the thermal conductivity and S_{rad} the radiative source term.

State equations

We have obtained five conservation equations: one for mass, three for momentum (as d is taken to be three) and one for enthalpy. There are seven unknowns: ρ, u_j $(j = 1, 2, 3)$, p, μ and e. A sufficient number of equations is obtained by two additional equations of state, see Wesseling [32] for more details.

General form of the conservation equations

The general form of the conservation equations of mass, momentum and enthalpy reads

$$\frac{\partial(\rho\phi)}{\partial t} + \frac{\partial F_j}{\partial x_j} = S_\phi, \tag{2.4}$$

where F_j is the total flux of quantity ϕ in the x_j direction. The total flux is the sum of the convective and the diffusive flux:

$$F_j = F_j^{con} + F_j^{diff} = \rho u_j \phi - \Gamma \frac{\partial \phi}{\partial x_j}, \tag{2.5}$$

with Γ the effective transport coefficient. For example for the mass conservation equation $\phi = 1$, $\Gamma = 0$ and $S_\phi = 0$.

2.2 The convection-diffusion equation

The convection-diffusion equation is given by (2.4) and (2.5). Taking $\rho = 1$ and renaming S_ϕ by q and ϕ by φ, the convection-diffusion equation can be written as

$$\frac{\partial \varphi}{\partial t} + (u_\alpha \varphi)_{,\alpha} - (\Gamma \varphi_{,\alpha})_{,\alpha} = q, \quad x \in \Omega \subset \mathbb{R}^d,\ 0 < t \leq T, \tag{2.6}$$

with T a given time. The equation is assumed to be linear, with φ the only unknown.
The dimensionless form of (2.6) is

$$\frac{\partial \varphi}{\partial t} + (u_\alpha \varphi)_{,\alpha} - (\varepsilon \varphi_{,\alpha})_{,\alpha} = q, \tag{2.7}$$

where $\varepsilon = Pe^{-1}$ with Pe the Péclet number, given by

$$Pe = \frac{UL}{\Gamma}, \tag{2.8}$$

with U and L typical velocity and length scales. For $Pe \gg 1$ convection dominates, for $Pe \ll 1$ diffusion dominates.

Initial and boundary conditions

At time $t = 0$ the following initial condition is given:

$$\varphi(0, \mathbf{x}) = \varphi_0(\mathbf{x}), \quad \mathbf{x} \in \Omega.$$

In order for the convection-diffusion equation together with its boundary conditions (BCs) to be a well-posed problem, the BCs have to meet two requirements: they have to be chosen such that the problem has a unique solution (because the convection-diffusion equation is a second order equation this means that one BC has to be prescribed at each part of the domain Ω). Also, they have to be

chosen such that small perturbations do not cause large changes in the solution. Suitable BCs for $\varepsilon \ll 1$ are given by (Wesseling, [32]):

$$\begin{aligned}
\varphi(t,\mathbf{x}) &= f_{in}(t,\mathbf{x}), \quad \mathbf{x} \in \partial\Omega_{in} \quad \text{(Dirichlet)} \\
\varphi(t,\mathbf{x}) &= f_{out}(t,\mathbf{x}), \quad \mathbf{x} \in \partial\Omega_{out} \quad \text{(Dirichlet), or} \\
\frac{\partial\varphi(t,\mathbf{x})}{\partial n} &= g_{out}(t,\mathbf{x}), \quad \mathbf{x} \in \partial\Omega_{out} \quad \text{(Neumann), or} \\
\frac{\partial\varphi(t,\mathbf{x})}{\partial n} + \varphi(t,\mathbf{x}) &= h_{out}(t,\mathbf{x}), \quad \mathbf{x} \in \partial\Omega_{out} \quad \text{(Robin),}
\end{aligned}$$

where n is the outward unit normal on the boundary $\partial\Omega$, $\partial\Omega_{in}$ is the inflow boundary (where $u_j n_j < 0$) and $\partial\Omega_{out}$ the outflow boundary ($u_j n_j > 0$).

2.2.1 The computing grid

In this chapter we restrict ourselves to the two-dimensional case for sake of simplicity. Generalization to three dimensions is straightforward. We will use a cell-centered uniform grid, see Figure 2.1. On the upper- and left side are the boundaries. The rectangular domain $\Omega = L_1 \times L_2$ is divided in rectangular cells of size $h_1 \times h_2$. The computational grid G is defined by:

$$\begin{aligned}
G = \{\mathbf{x} \in \Omega : \mathbf{x} = \mathbf{x_j} = (\mathbf{j} - \mathbf{p})\mathbf{h}, \mathbf{j} = (j_1, j_2), \mathbf{p} = \left(\frac{1}{2}, \frac{1}{2}\right), \\
\mathbf{h} = (h_1, h_2), j_\alpha = 1, 2, ..., n_\alpha, h_\alpha = 1/n_\alpha\}.
\end{aligned}$$

The cell with center $\mathbf{x_j}$ is called $\Omega_\mathbf{j}$. Define

$$\mathbf{e}_1 \equiv (1/2, 0) \text{ and } \mathbf{e}_2 \equiv (0, 1/2).$$

The value of a quantity φ in $\mathbf{x_j}$ is denoted by $\varphi_\mathbf{j}$, and $\varphi_{\mathbf{j}+\mathbf{e}_1}$ is located at a cell face, namely at

$$\mathbf{x}_{\mathbf{j}+\mathbf{e}_1} = ((j_1 + 1/2)h_1, j_2 h_2).$$

The cell at the 'east' side of $\Omega_\mathbf{j}$ is designated by $\Omega_{\mathbf{j}+2\mathbf{e}_1}$.

2.2.2 Discretization

In order to solve the convection-diffusion equation (2.7) numerically, the equation has to be discretized. In this subsection space discretization is described. Time discretization can be found in for example Wesseling [32]

Space discretization

We rewrite (2.7) as

$$\frac{\partial\varphi}{\partial t} + L\varphi = q, \qquad L\varphi = (u_\alpha\varphi)_{,\alpha} - (\varepsilon\varphi_{,\alpha})_{,\alpha}. \tag{2.9}$$

We will now integrate (2.9) over a cell $\Omega_\mathbf{j}$. Integrating the first term in the left-hand side (LHS) of (2.9) and using the midpoint rule gives:

$$\int_{\Omega_\mathbf{j}} \frac{\partial\varphi}{\partial t} d\Omega \approx h_1 h_2 \frac{d\varphi_j}{dt}.$$

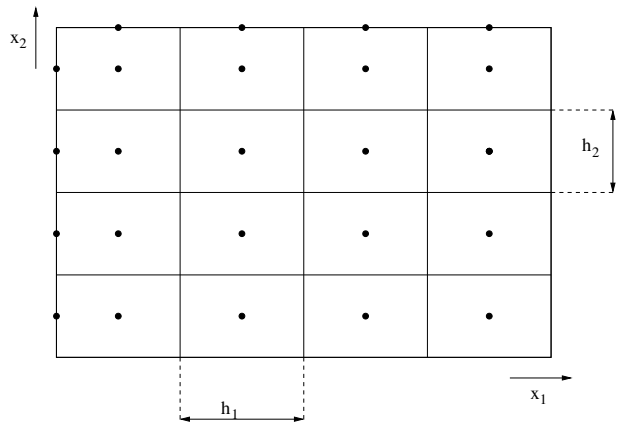

Fig. 2.1: A cell-centered uniform grid

Integrating the second term in the LHS of (2.9) using the divergence theorem gives:

$$\begin{aligned}\int_{\Omega_{\mathbf{j}}} L\varphi d\Omega = L_h\phi_{\mathbf{j}} &= \left[\int_{\mathbf{x_{j+e_1+e_2}}}^{\mathbf{x_{j-e_1+e_2}}} - \int_{\mathbf{x_{j+e_1-e_2}}}^{\mathbf{x_{j-e_1-e_2}}}\right](u_1\varphi - \varepsilon\varphi_{,1})dx_2 \\ &+ \left[\int_{\mathbf{x_{j-e_1+e_2}}}^{\mathbf{x_{j+e_1+e_2}}} - \int_{\mathbf{x_{j-e_1-e_2}}}^{\mathbf{x_{j+e_1-e_2}}}\right](u_2\varphi - \varepsilon\varphi_{,2})dx_1 \\ &= F^1|_{\mathbf{j-e_1}}^{\mathbf{j+e_1}} + F^2|_{\mathbf{j-e_2}}^{\mathbf{j+e_2}}.\end{aligned}$$

L_h is a discrete operator and F^α is the numerical flux. The integrals in the above equation will be approximated by the midpoint rule. The diffusive flux, for example at the 'east' side, can be approximated by

$$\int_{\mathbf{x_{j+e_1-e_2}}}^{\mathbf{x_{j+e_1+e_2}}} (-\varepsilon\varphi_{,1})dx_2 \approx -\varepsilon(\varphi_{\mathbf{j+2e_1}} - \varphi_{\mathbf{j}})\frac{h_2}{h_1}.$$

The approximation of the convective flux is described in Subsection 2.2.3. Integrating the source term in the right-hand side (RHS) of equation (2.9) yields

$$\int_{\Omega_{\mathbf{j}}} q d\Omega \approx h_1 h_2 q_{\mathbf{j}} = \hat{q}_{\mathbf{j}}.$$

2.2.3 Interpolation practices

Central and upwind discretization

There are various interpolation methods for approximating the convective flux. One way is to use a *central difference scheme* (CDS), which reads for a uniform grid:

$$(u\varphi)_{\mathbf{j+e_\alpha}} \approx \frac{1}{2}u_{\mathbf{j+e_\alpha}}(\varphi_{\mathbf{j}} + \varphi_{\mathbf{j+2e_\alpha}}).$$

The CDS is $\mathcal{O}(h_\alpha^2)$ accurate.
Another way is to use an *upwind difference scheme* (UDS), and approximate $u\varphi$ by the value of the node upstream:

$$(u\varphi)_{\mathbf{j}+\mathbf{e}_\alpha} \approx \frac{1}{2}(u_{\mathbf{j}+\mathbf{e}_\alpha} + |u_{\mathbf{j}+\mathbf{e}_\alpha}|)\varphi_{\mathbf{j}} + \frac{1}{2}(u_{\mathbf{j}+\mathbf{e}_\alpha} - |u_{\mathbf{j}+\mathbf{e}_\alpha}|)\varphi_{\mathbf{j}+2\mathbf{e}_\alpha}.$$

The UDS is $\mathcal{O}(h_\alpha)$ accurate.

Wiggles

Discretization of the convection-diffusion equation leads to a FV scheme. If this scheme is of the so-called positive type, which can be verified by the scheme's stencil, a discrete maximum principle can be formulated, which can give us a priori information. By the sign of the source term this principle tells us if the exact solution has a local maximum or minimum. If the exact solution has no local extrema, "wiggles" (oscillations) in a numerical solution are not physical but must be a numerical artefact. More information on this subject can be found in Wesseling [32].
One can verify that for the convection-diffusion equation with a constant velocity field the UDS is positive for all Péclet numbers and satisfies the discrete maximum principle, so with this scheme no wiggles occur. On the other hand, one can verify that the CDS may introduces wiggles if

$$p_{\mathbf{j}+\mathbf{e}_\alpha} = \frac{|u_{\mathbf{j}+\mathbf{e}_\alpha}|h_{j_\alpha}}{\varepsilon} \geq 2, \tag{2.10}$$

where the dimensionless number p is called the mesh Péclet number.

Hybrid scheme

In order to have a discretization scheme that does not introduce wiggles, an option is to choose an UDS. However, this scheme is only first order accurate and it introduces numerical diffusion. A solution to this problem is to use the UDS only in regions where $p \geq 2$ and the central scheme elsewhere. This is called the *hybrid difference scheme* (HDS) and can be done as follows:

$$(u\varphi)_{\mathbf{j}+\mathbf{e}_\alpha} \approx s(p_{\mathbf{j}+\mathbf{e}_\alpha})(u\varphi)_{uds,\mathbf{j}+\mathbf{e}_\alpha} + (1 - s(p_{\mathbf{j}+\mathbf{e}_\alpha}))(u\varphi)_{cds,\mathbf{j}+\mathbf{e}_\alpha},$$

where $s(p_{\mathbf{j}+\mathbf{e}_\alpha})$ is a switch function with the mesh Péclet number defined by (2.10). For a more detailed description of the HDS the reader is referred to Wesseling [32].
For iterative convergence in nonlinear cases it is beneficial if $s(p)$ switches smoothly between 0 and 1. Furthermore, one can show that the HDS is $\mathcal{O}(h_\alpha)$ accurate.

2.2.4 Convergence, consistency and stability

A numerical scheme is only useful if it is convergent. How this can be accomplished will now be discussed briefly.

Global and local truncation error

Truncation errors are errors that are caused by truncation (to truncate means to shorten by cutting off) of an infinite process. The process we consider is where the maximum mesh size goes to zero. In the numerical approximation of differential equations rounding errors are usually much smaller than truncation errors, so we assume a zero rounding error.
The *global truncation error* $e^{(n)}$ is defined by

$$e^{(n)} = \varphi_e^{(n)} - \varphi^{(n)},$$

where $\varphi_e^{(n)}$ denotes the algebraic vector with elements of the exact solution evaluated at time t_n and $\varphi^{(n)}$ denotes the algebraic vector with elements of the numerical approximation evaluated at time t_n. The *local truncation error* $r^{(n)}$ of the discrete operator L_h is defined by

$$r^{(n)} = L_h e^{(n)}.$$

Convergence

A numerical scheme is convergent if the global truncation error satisfies

$$\lim_{h,\tau \downarrow 0} e_{j_h}^{(T/\tau)} = 0, \quad x_{j_h} \text{fixed}.$$

τ is the time step, it is assumed that τ and the spatial mesh sizes belong to a decreasing sequence such that T/τ is an integer and $\mathbf{x}_{j_h}$ is a grid point in the spatial grid. Convergence means that the exact solution can be approximated arbitrarily closely at a fixed time and position by decreasing τ and refining the grid.

Consistency

A numerical scheme is consistent if

$$\|r^{(n)}\| \downarrow 0$$

for $n = 1, 2, ..., T/\tau$ and $h \downarrow 0$. Consistency means that for any point in the domain the truncation error goes to zero as the mesh size goes to zero.

Stability

Consistency does not imply convergence. In addition, stability is required. Let $\delta^{(0)}$ be a hypothetical arbitrary perturbation of $\varphi^{(0)}$. The resulting perturbation of $\varphi^{(n)}$ is called $\delta^{(n)}$. Now a numerical scheme is called stable if $\delta^{(n)}$ remains bounded as $n \to \infty$, for all $\delta^{(0)}$. Two useful definitions of stability are zero-stability and absolute-stability.
A scheme is called *zero-stable* if there exists a bounded function $C(T)$ and a function $\tau_0(h)$ such that for arbitrary $\delta^{(0)}$

$$\|\delta^{(T/\tau)}\|_h \leq C(T)\|\delta^{(0)}\|_h$$

for all $\tau \leq \tau_0(h)$, all $h \leq h_0$ for some fixed h_0 and $\|x\|_h = (|x_1|^h + \ldots + |x_n|^h)^{1/h}$, $h \geq 1$.
A scheme is called *absolutely-stable* if there exists a constant C and a function $\tau_0(h)$ such that

$$\|\delta^{(n)}\|_h \leq C\|\delta^{(0)}\|_h$$

for h fixed, all $n > 0$ and all $\tau \leq \tau_0(h)$. Absolute stability differs from zero-stability by the fact that h is fixed.

It is favourable to have zero-stability because when the scheme is consistent (which is often not so difficult to prove) we can apply Lax's equivalence theorem, which reads

$$\begin{aligned} \text{zero-stability + consistency} &\Rightarrow \text{convergence,} \\ \text{zero-stability} &\Leftarrow \text{convergence.} \end{aligned}$$

The main difficulty is to prove zero-stability.

2.3 The incompressible Navier-Stokes equations

The incompressible Navier-Stokes equations are given by (2.2). The dimensionless form is

$$\frac{\partial(\rho u_\alpha)}{\partial t} + (\rho u_\alpha u_\beta)_{,\beta} = -p_{,\alpha} + \sigma_{\alpha\beta,\beta} + f_\alpha, \tag{2.11}$$

with

$$\sigma_{\alpha\beta} = Re^{-1}(u_{\alpha,\beta} + u_{\beta,\alpha}),$$

f a body force and the Reynolds number Re given by

$$Re = \frac{\rho_0 UL}{\mu}.$$

Initial conditions

For the momentum equations the following initial condition is required:

$$u_\alpha(0, \mathbf{x}) = w_\alpha(\mathbf{x}),$$

with $w_\alpha(\mathbf{x})$ the velocity at $t = 0$.

No-slip condition

Viscous fluids cling to solid surfaces, this is called the *no-slip condition*. At a solid surface we have

$$u_\alpha(t, \mathbf{x}) = v_\alpha(t, \mathbf{x}),$$

with $v_\alpha(t, \mathbf{x})$ the local wall velocity.

Free surface conditions

At a free surface the tangential stress components are zero. We consider the special case where the free surface is fixed at $y = a =$ constant. In that case the normal velocity and the tangential stress $\sigma_{\alpha\beta}$, $\alpha \neq \beta$ are zero:

$$v(t, \mathbf{x}, a) = 0, \quad u_y(t, \mathbf{x}, a) = 0.$$

Inflow conditions

Because the momentum equations resemble convection-diffusion equations for the velocities, we prescribe Dirichlet conditions at an inflow boundary, see Section 2.2. If $x = a$ is an inflow boundary, we prescribe

$$u_\alpha(t, a, y) = U_\alpha(t, y).$$

Outflow conditions

At an outflow boundary, often not enough physical information is available on which to base a sufficient number of BCs; usually only the pressure is known. Because of the resemblance of (2.11) to the convection-diffusion equation, it is plausible that when $Re \gg 1$, 'wrong' information generated by an artificial boundary condition propagates upstream only over a distance of $\mathcal{O}(Re^{-1})$. This can be shown by applying singular perturbation analysis. In order to avoid numerical wiggles it is advisable to choose as artificial outflow condition a homogeneous Neumann condition for the tangential velocity. For an outflow boundary at $x = a$ this gives:

$$p(t, a, y) = p_\infty, \quad \rho v_x(t, a, y) = 0.$$

More information on this subject can be found in Wesseling [32].

2.3.1 Discretization on a colocated grid

Colocated and staggered grids

There are two ways to arrange the unknowns on a grid: colocated arrangement and staggered arrangement, see Figure 2.2. When all discrete unknowns reside in one grid point (often the cell centers), the grid is called a colocated grid. When the pressure is located in the cell centers and the velocity components are located at the cell face centers, the grid is called a staggered grid.
We will restrict ourselves to a colocated grid and to incompressible flow with constant density $\rho = 1$.

Discretization of the continuity equation

An incompressible flow means that $u_{\alpha,\alpha} = 0$. FV discretization gives

$$\int_{\Omega_\mathbf{j}} u^\alpha_{,\alpha} d\Omega \cong h_2 u^1 |^{\mathbf{j}+\mathbf{e}_1}_{\mathbf{j}-\mathbf{e}_1} + h_1 u^2 |^{\mathbf{j}+\mathbf{e}_2}_{\mathbf{j}-\mathbf{e}_2} = 0. \tag{2.12}$$

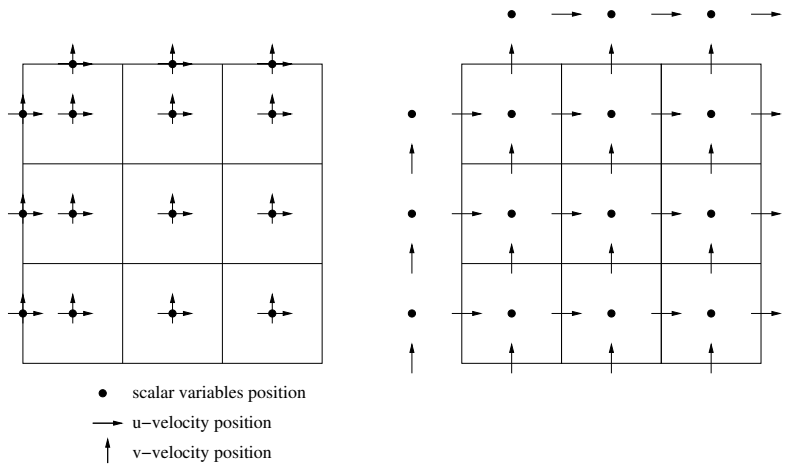

Fig. 2.2: A colocated grid (left) and a staggered grid (right)

The cell face values in (2.12) need to be interpolated in terms of cell center values. The straightforward way to do this is to use linear interpolation, resulting in

$$h_2 u^1|_{\mathbf{j}-2\mathbf{e}_1}^{\mathbf{j}+2\mathbf{e}_1} + h_1 u^2|_{\mathbf{j}-2\mathbf{e}_2}^{\mathbf{j}+2\mathbf{e}_2} = 0.$$

Discretization of the momentum equation

Taking $f = 0$ in (2.11), we have to discretize

$$\frac{\partial u^\alpha}{\partial t} + F^{\alpha\beta}_{,\beta} + p_{,\alpha} = 0, \quad F^{\alpha\beta} = u^\alpha u^\beta - \sigma^{\alpha\beta}.$$

FV discretization gives

$$\int_{\Omega_{\mathbf{j}}} \{\frac{\partial u^\alpha}{\partial t} + F^{\alpha\beta,\beta} + p_{,\alpha}\} d\Omega \cong h_1 h_2 \frac{du^\alpha_{\mathbf{j}}}{dt} + h_2 F^{\alpha 1}|_{\mathbf{j}-\mathbf{e}_1}^{\mathbf{j}+\mathbf{e}_1} + h_1 F^{\alpha 2}|_{\mathbf{j}-\mathbf{e}_2}^{\mathbf{j}+\mathbf{e}_2} + h_\gamma p|_{\mathbf{j}-\mathbf{e}_\alpha}^{\mathbf{j}+\mathbf{e}_\alpha} = 0, \tag{2.13}$$

with $\gamma \neq \alpha$. The cell face values have to be interpolated between cell center values. For a staggered grid, this can be done quite straightforward, for the pressure as follows:

$$p_{\mathbf{j}+\mathbf{e}_\alpha} = \frac{1}{2}(p_{\mathbf{j}} + p_{\mathbf{j}+2\mathbf{e}_\alpha}). \tag{2.14}$$

The interpolation for the pressure for the colocated grid will be described later, see equations (2.19)–(2.22). The viscous stress $\sigma^{\alpha\beta}$ is approximated using

$$\begin{aligned} (Re^{-1} u^\alpha_{,1})_{\mathbf{j}+\mathbf{e}_1} &\cong Re^{-1}_{\mathbf{j}+\mathbf{e}_1} (u^\alpha_{\mathbf{j}+2\mathbf{e}_1} - u^\alpha_{\mathbf{j}})/h_1, \\ (Re^{-1} u^\alpha_{,2})_{\mathbf{j}+\mathbf{e}_1} &\cong \frac{1}{4} Re^{-1}_{\mathbf{j}+\mathbf{e}_1} \left(u^\alpha|_{\mathbf{j}-2\mathbf{e}_2}^{\mathbf{j}+2\mathbf{e}_2} + u^\alpha|_{\mathbf{j}+2\mathbf{e}_1-2\mathbf{e}_2}^{\mathbf{j}+2\mathbf{e}_1+2\mathbf{e}_2} \right)/h_2, \end{aligned} \tag{2.15}$$

etcetera. For the inertia terms we may use

$$(u^\alpha u^\beta)_{\mathbf{j}+\mathbf{e}_\beta} \cong \frac{1}{2}(u^\alpha u^\beta)_{\mathbf{j}} + \frac{1}{2}(u^\alpha u^\beta)_{\mathbf{j}+2\mathbf{e}_\beta}, \tag{2.16}$$

corresponding to second order central discretization.

Spurious checkerboard modes

The preceding discretization is analogous to what was done for the convection-diffusion equation. But now, using the CDS to approximate the terms in equations (2.12) and (2.13) causes a spurious checkerboard pattern. Assuming Re is constant and neglecting boundary conditions, one can show that

$$u^\alpha_{\mathbf{j}} = (-1)^{j_1+j_2} \exp\{-\frac{12}{Re}(h_1^{-2} + h_2^{-2})t\}, \quad p = (-1)^{j_1+j_2} \tag{2.17}$$

with homogeneous boundary conditions, is a solution of (2.13)-(2.16). This shows that if $Re \gg 1$ the checkerboard pattern (2.17) is damped slowly for the velocity, but not for the pressure. One way to avoid checkerboard patterns is to use a staggered grid instead of a colocated grid. Another option is to use the pressure-weighted interpolation method, which will be discussed next.

Pressure-weighted interpolation method

The pressure-weighted interpolation (PWI) method (or Rhie & Chow interpolation after its inventors) is a colocated discretization method of accuracy $\mathcal{O}(h_1^2 + h_2^2)$. The idea is to prevent checkerboard oscillations by perturbing the continuity equation with pressure terms. The continuity equation is discretized as in (2.12), with the cell face velocities evaluated as follows:

$$u^\alpha_{\mathbf{j}+\mathbf{e}_\alpha} = \frac{1}{2}(u^\alpha_{\mathbf{j}} + u^\alpha_{\mathbf{j}+2\mathbf{e}_\alpha}) + (\frac{h_\beta}{4a^\alpha_j}\Delta^\alpha p)|^{\mathbf{j}+2\mathbf{e}_\alpha}_{\mathbf{j}} \quad \text{(no summation)}, \tag{2.18}$$

where $\Delta^\alpha p_{\mathbf{j}} = p_{\mathbf{j}+2\mathbf{e}_\alpha} - 2p_{\mathbf{j}} + p_{\mathbf{j}-2\mathbf{e}_\alpha}$, $\beta \neq \alpha$ and a^α_j equals the negative sum of the coefficients of u^α_k, $k \neq j$ in equation (2.13). The basic mathematical principle behind (2.18) is that a small regularizing term is added (the second term in the RHS) that excludes spurious modes.
Substitution of (2.18) in (2.12) results in the following discretization of $u^\alpha_{,\alpha}$ with the PWI method:

$$\begin{aligned} & h_2 u^1|^{\mathbf{j}+2\mathbf{e}_1}_{\mathbf{j}-2\mathbf{e}_1} + h_1 u^2|^{\mathbf{j}+2\mathbf{e}_2}_{\mathbf{j}-2\mathbf{e}_2} \\ + \quad & h_2^2\{(\frac{1}{2a^1}\Delta^1 p)_{\mathbf{j}+2\mathbf{e}_1} - (\frac{1}{a^1}\Delta^1 p)_{\mathbf{j}} + (\frac{1}{2a^1}\Delta^1 p)_{\mathbf{j}-2\mathbf{e}_1}\} \\ + \quad & h_1^2\{(\frac{1}{2a^2}\Delta^2 p)_{\mathbf{j}+2\mathbf{e}_2} - (\frac{1}{a^2}\Delta^2 p)_{\mathbf{j}} + (\frac{1}{2a^2}\Delta^2 p)_{\mathbf{j}-2\mathbf{e}_2}\} = 0. \end{aligned}$$

Boundary conditions and pressure

A disadvantage of the PWI method is that it requires some further specification of conditions for p at boundaries, beyond what is given for the differential equations. Let $(j_1 = 1, j_2)$, so that the 'west' face of $\Omega_{\mathbf{j}}$ is part of the boundary.

In (2.13), $p_{\mathbf{j}-\mathbf{e}_1}$ occurs, referring to a grid point at the boundary. This value is approximated by extrapolation from the interior,

$$p_{\mathbf{j}-\mathbf{e}_1} = \frac{3}{2}p_{\mathbf{j}} - \frac{1}{2}p_{\mathbf{j}+2\mathbf{e}_1}. \tag{2.19}$$

When the pressure distribution at the boundaries needs to be computed this can be obtained by the method by Choi et al. [2]. First the pressure correction is calculated:

$$p'_{\mathbf{j}-\mathbf{e}_1} = p'_{\mathbf{j}} + \frac{(u^*_{\mathbf{j}} - u_{\mathbf{j}-\mathbf{e}_1})h_2 + \frac{1}{2}(u_{\mathbf{j}-\mathbf{e}_2} - u_{\mathbf{j}+\mathbf{e}_2})h_1}{D^1_j h_2}, \tag{2.20}$$

where $D^1_j = \frac{\alpha_j h_2}{a^1_j}$, with α_j a relaxation factor and $p'_{\mathbf{j}}$ known from a previous iteration level. $u^*_{\mathbf{j}}$ is given by

$$u^*_{\mathbf{j}} = u_{\mathbf{j}} - D^1_j(p'_{\mathbf{j}} - p'_{\mathbf{j}-\mathbf{e}_1}). \tag{2.21}$$

After the pressure correction is calculated, the pressure can be obtained by the following formula:

$$p_{\mathbf{j}-\mathbf{e}_1} = p^{l-1}_{\mathbf{j}-\mathbf{e}_1} + \alpha_j p'_{\mathbf{j}-\mathbf{e}_1}. \tag{2.22}$$

$p^{l-1}_{\mathbf{j}-\mathbf{e}_1}$ is the pressure at the boundary at a previous iteration level.

Summary of equations

After spatial discretization, a system of ordinary differential equations is obtained. An algebraic vector $\mathbf{u}$ contains all unknown velocity components and an algebraic vector $\mathbf{p}$ contains all pressure unknowns. These equations can be written as

$$\begin{aligned} \frac{d\mathbf{u}}{dt} + N(\mathbf{u})\mathbf{u} + G\mathbf{p} &= f(t), \\ D\mathbf{u} + C\mathbf{p} &= g(t), \end{aligned} \tag{2.23}$$

where N is a nonlinear algebraic operator arising from the discretization of the inertia and viscous terms, G is a linear algebraic operator representing the discretization of the pressure gradient, D and C are linear algebraic operators corresponding to the velocity terms and the pressure terms, respectively, in the PWI discretization of the continuity equation. f and g are known source terms arising from the boundary conditions and body forces. The system (2.23) contains both differential and algebraic systems and is therefore called a differential-algebraic system (DAS).

3. ITERATIVE SOLUTION METHODS

Problems coming from discretized PDEs lead in general to large sparse systems of equations. In that case, direct solution methods can be very impractical. Consider the system $A\mathbf{x} = \mathbf{b}$ that has to be solved for $\mathbf{x}$. Direct solution methods, for example the Gaussian elimination method, use the LU decomposition of the matrix A. If A is large and sparse, L and U can be dense. This is especially the case for 3D problems. So a considerable amount of memory is required and the solution costs many floating point operations.
Therefore, iterative methods are used. These methods generate a sequence of approximate solutions $\{\mathbf{x}^{(k)}\}$. The use of iterative solution methods is especially very attractive in time-dependent and nonlinear problems.

In this chapter, several iterative methods are described. In the first three sections, iterative methods for solving linear equations are discussed: in Section 3.1 basic iterative methods are treated. In Section 3.2 Krylov subspace methods and the preconditioning of these methods are discussed. Deflated Krylov subspace methods are treated in Section 3.3. How the discretized incompressible Navier-Stokes equations can be solved is described in Section 3.4.

3.1 Basic iterative solution methods

This section reviews some basic iterative methods (BIMs). A survey can be found in for example Wesseling [32], Vuik [23], Saad [14] and Van der Vorst [22]. Given an $n \times n$ real matrix A and a real n-vector $\mathbf{b}$, the problem considered is:

$$A\mathbf{x} = \mathbf{b} \tag{3.1}$$

Equation (3.1) is a *linear system*, A is the *coefficient matrix*, $\mathbf{b}$ is the *right-hand side vector* and $\mathbf{x}$ is the *vector of unknowns*.
The basic idea behind iterative methods for the solution of a linear system is: starting from a given $\mathbf{x}^{(k)}$, obtain a better approximation $\mathbf{x}^{(k+1)}$ of $\mathbf{x}$ in a cheap way. Note that $\mathbf{b} - A\mathbf{x}^{(k)}$ is small if $\mathbf{x}^{(k)}$ is close to $\mathbf{x}$. This motivates the iteration process

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + M^{-1}(\mathbf{b} - A\mathbf{x}^{(k)}). \tag{3.2}$$

It is easy to verify that if this process converges, $\mathbf{x}^{(k)}$ is a possible solution. The choice of M is crucial in order to obtain a fast converging iterative method. Rewriting of (3.2) leads to:

$$M\mathbf{x}^{(k+1)} = N\mathbf{x}^{(k)} + \mathbf{b}, \tag{3.3}$$

where the matrix N is given by $N = M - A$. The formula $A = M - N$ is also known as the splitting of A.

Some well-known BIMs are the Gauss-Jacobi method, the Gauss-Seidel method and the Strongly Implicit Procedure (SIP). The SIP method is often used in X-stream and is described in, for example, Ferziger & Perić [3].

Convergence

The error $\mathbf{e}^{(k)} = \mathbf{x} - \mathbf{x}^{(k)}$ satisfies

$$\mathbf{e}^{(k+1)} = B\mathbf{e}^{(k)},$$

with $B = M^{-1}N$, called the *iteration matrix*. We have

$$\|\mathbf{e}^{(k)}\| \leq \|B^k\|\|\mathbf{e}^{(0)}\|. \tag{3.4}$$

The spectral radius of a matrix B is defined by

$$\rho(B) = max\{|\lambda|, \text{where } \lambda \in \text{spectrum of } B\}$$

and satisfies

$$\rho(B) = \lim_{k\to\infty} \|B^k\|^{1/k}.$$

There is convergence if and only if

$$\rho(B) < 1.$$

The smaller $\rho(B)$, the faster the convergence. Then

$$\|B^k\| = \rho(B)^k, \quad \text{for} \quad k \to \infty,$$

and (3.4) becomes

$$\|\mathbf{e}^{(k)}\| \leq \rho(B)^k\|\mathbf{e}^{(0)}\|.$$

Starting vectors

Iterative solution methods solving $A\mathbf{x} = \mathbf{b}$ start with a given vector $\mathbf{x}^{(0)}$. Here we shall give some ideas how to choose a good starting vector $\mathbf{x}^{(0)}$. These choices depend on the problem to be solved. If no further information is available one always starts with $\mathbf{x}^{(0)} = 0$.
The solution of a nonlinear problem is in general approximated by the solution of a number of linear systems. In such a problem the final solution of the iterative method used to solve a linear system can be used as a starting solution for the iterative method used to solve the next linear system.

Termination criteria

In general, the iterative method should be stopped if the approximate solution is accurate enough. A good termination criterion is very important for an iterative method, because if the criterion is too weak the approximate solution is useless,

whereas if the criterion is too severe the iterative method never stops or costs too much work.
An important quality a termination criterion should have is that it has to be scaling invariant. An example of a good criterion is:

$$\frac{\|\mathbf{b} - A\mathbf{x}^{(k)}\|_2}{\|\mathbf{b}\|_2} \leq \varepsilon, \tag{3.5}$$

with

$$\|\mathbf{x}\|_m = (x_1^m + \ldots + x_n^m)^{1/m}. \tag{3.6}$$

3.2 Krylov subspace methods

Define the computing work W to solve the linear system (3.1) as

$$W = \mathcal{O}(\mathcal{N}^\alpha), \tag{3.7}$$

with $\mathcal{N}$ the total number of equations and α a certain number. It can be shown that for BIMs, for discretization of elliptic PDEs, in general $\alpha \approx 2$, the same as for the 'best' direct methods. Fortunately, BIMs can be accelerated. This can be done by multigrid acceleration and Krylov subspace acceleration. Multigrid methods bring α down to 1 (the ideal case), but are in general more difficult to implement. For a survey on multigrid methods see for example Hackbusch [6] or Wesseling [31]. Krylov subspace methods come close to $\alpha = 1$ and are much easier to implement. A survey on Krylov subspace methods can be found in for example Saad [14], Van der Vorst [22] or Vuik [23]. In this section Krylov subspace methods are discussed.

In the BIMs iterates are computed by the following recursion:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + M^{-1}(\mathbf{b} - A\mathbf{x}^{(k)}) = \mathbf{x}^{(k)} + M^{-1}\mathbf{r}^{(k)}. \tag{3.8}$$

Writing out the first steps of such a process we obtain:

$$\begin{aligned}
\mathbf{x}^{(0)} &, \\
\mathbf{x}^{(1)} &= \mathbf{x}^{(0)} + M^{-1}\mathbf{r}^{(0)}, \\
\mathbf{x}^{(2)} &= \mathbf{x}^{(1)} + M^{-1}\mathbf{r}^{(1)} = \mathbf{x}^{(0)} + M^{-1}\mathbf{r}^{(0)} + M^{-1}(\mathbf{b} - A\mathbf{x}^{(0)} - AM^{-1}\mathbf{r}^{(0)}) \\
&= \mathbf{x}^{(0)} + 2M^{-1}\mathbf{r}^{(0)} - M^{-1}AM^{-1}\mathbf{r}^{(0)}, \\
&\vdots
\end{aligned}$$

This implies that

$$\mathbf{x}^{(k)} \in \mathbf{x}^{(0)} + \text{span}\{M^{-1}\mathbf{r}^{(0)}, M^{-1}A(M^{-1}\mathbf{r}^{(0)}), \ldots, (M^{-1}A)^{k-1}(M^{-1}\mathbf{r}^{(0)})\}. \tag{3.9}$$

The subspace $K^{(i)}(A; \mathbf{r}^{(0)}) = \text{span}\{\mathbf{r}^{(0)}, A\mathbf{r}^{(0)}, \ldots, A^{i-1}\mathbf{r}^{(0)}\}$ is called the Krylov subspace of dimension i corresponding to matrix A and initial residual $\mathbf{r}^{(0)}$. An $\mathbf{x}^{(i)}$ calculated by a BIM is an element of $\mathbf{x}^{(0)} + K^{(i)}(M^{-1}A; M^{-1}\mathbf{r}^{(0)})$. One can work out that $\mathbf{x}^{(m)} - \mathbf{x}^{(0)} \in K^{(m)}(A, \mathbf{r}^{(0)})$. Methods that look for optimal

approximations to $\mathbf{x}-\mathbf{x}^{(0)}$ in $K^{(m)}(A, \mathbf{r}^{(0)})$ are called Krylov subspace methods.

Convergence

It can be shown that for a symmetric positive definite (SPD) matrix A the convergence behaviour of Krylov subspace methods depends strongly on the eigenvalue distribution of the coefficient matrix. The condition number κ of an SPD matrix A is given by

$$\kappa(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)}. \tag{3.10}$$

A smaller condition number means faster convergence.

Preconditioning

Krylov subspace methods as explained above are well founded theoretically, but they are all likely to suffer from slow convergence for problems which arise from typical applications such as CFD. Both the efficiency and robustness of iterative techniques can be improved by using preconditioning. Preconditioning is simply a means of transforming the original linear system into one which has the same solution, but which is likely to be easier to solve with an iterative solver. In general, the reliability of iterative techniques, when dealing with various applications, depends much more on the quality of the preconditioner than on the particular Krylov subspace method used.

A preconditioner is a matrix that transforms the system such that the transformed coefficient matrix has a more favourable spectrum. Consider the preconditioner M. The transformed system is then given by

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}. \tag{3.11}$$

The matrix M has the following requirements: $M^{-1}\mathbf{y} = \mathbf{z}$ has to be inexpensive to solve for y and the eigenvalues of $M^{-1}A$ should be clustered around 1.
In literature the term preconditioner has a double meaning. Besides referring to the preconditioning matrix, the entire iterative method is often called a preconditioner.

The Preconditioned Conjugate Gradient (PCG) method

In the special case that A is SPD, using the PCG method is not expensive. More information about this method can be found in for example Vuik [23], Wesseling [32] and Saad [14]. The PCG method is given by the following algorithm:

Algorithm 3.1 (PCG)
Given an SPD $n \times n$ matrix A, an n-vector $\mathbf{b}$, an SPD preconditioner M and an initial guess $\mathbf{x}^{(0)}$, this algorithm solves the linear system $A\mathbf{x} = \mathbf{b}$.
$\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$
$\mathbf{z}^{(0)} = M^{-1}\mathbf{r}^{(0)}$
$\mathbf{p} = \mathbf{z}^{(0)}$

For $j = 1, \ldots,$ until convergence **do**:
$\alpha = (\mathbf{r}^{(j-1)}, \mathbf{z}^{(0)})/(A\mathbf{p}, \mathbf{p})$
$\mathbf{x}^{(j)} = \mathbf{x}^{(j-1)} + \alpha\mathbf{p}$
$\mathbf{r}^{(j)} = \mathbf{r}^{(j-1)} - \alpha A\mathbf{p}$
$\mathbf{z}^{(1)} = M^{-1}\mathbf{r}^{(j)}$
$\beta = (\mathbf{r}^{(j)}, \mathbf{z}^{(1)})/(\mathbf{r}^{(j-1)}, \mathbf{z}^{(0)})$
$\mathbf{p} = \mathbf{z}^{(1)} + \beta\mathbf{p}$
$\mathbf{z}^{(0)} = \mathbf{z}^{(1)}$
End for
$\mathbf{x} \approx \mathbf{x}^{(j)}$

Note that when implementing this algorithm, not everything has to be kept in memory. For example, in iteration j, only $\mathbf{r}^{(j-1)}$ and $\mathbf{r}^{(j)}$ are needed. There is no use keeping all the earlier vectors $\mathbf{r}^{(i)}$, $i < j - 1$.

Computational work, robustness and convergence

Orthonormalization is not required in the PCG method, in contrast to other Krylov methods, as we will see later. This makes the PCG method cheap and it requires little memory. Break-down does not occur in computing α and β, because A and M are SPD. Therefore the PCG algorithm is robust. Concerning convergence, one can show that

$$\|\mathbf{x} - \mathbf{x}^{(k)}\|_{M^{-1}A} \leq 2 \left(\frac{\sqrt{\kappa(M^{-1}A)} - 1}{\sqrt{\kappa(M^{-1}A)} + 1} \right)^k \|\mathbf{x} - \mathbf{x}^{(0)}\|_{M^{-1}A}, \tag{3.12}$$

where $\|\mathbf{x}\|_{M^{-1}A} = \sqrt{(M^{-1}A\mathbf{x}, \mathbf{x})}$.

The Preconditioned Generalized Conjugate Residual (PGCR) method

For a general matrix A the GCR Krylov subspace method can be used to solve the linear system $A\mathbf{x} = \mathbf{b}$. The preconditioned GCR method is given by the following algorithm, see for example Vuik, Frank & Segal [26].

Algorithm 3.2 (PGCR)
Given an $n \times n$ matrix A, an n-vector $\mathbf{b}$, a preconditioner M and an initial guess $\mathbf{x}^{(0)}$, this algorithm solves the linear system $A\mathbf{x} = \mathbf{b}$.
$\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$
For $j = 1, \ldots,$ until convergence **do**:
Solve $M\mathbf{v}^{(j)} = \mathbf{r}^{(j-1)}$
$\mathbf{q}^{(j)} = A\mathbf{v}^{(j)}$
$[\mathbf{q}^{(j)}, \mathbf{v}^{(j)}] = \mathbf{orthonorm}[\mathbf{q}^{(j)}, \mathbf{v}^{(j)}, \mathbf{q}^{(i)}, \mathbf{v}^{(i)}, i < j]$
$\gamma = (\mathbf{q}^{(j)}, \mathbf{r}^{(j-1)})$
$\mathbf{x}^{(j)} = \mathbf{x}^{(j-1)} + \gamma\mathbf{v}^{(j)}$
$\mathbf{r}^{(j)} = \mathbf{r}^{(j-1)} - \gamma\mathbf{q}^{(j)}$
End for
$\mathbf{x} \approx \mathbf{x}^{(j)}$

Note that the relation between $\mathbf{q}$ and $\mathbf{v}$ ($\mathbf{q} = A\mathbf{v}$) is very important and should be satisfied for all intermediate steps. The routine **orthonorm** refers to the orthonormalization process used. It is left open in the above algorithm because there are several ways to do this, see Frank & Vuik [4]. A common used method is the Modified Gram-Schmidt algorithm:

$[\mathbf{q}, \mathbf{v}] = \mathbf{orthonorm}[\mathbf{q}, \mathbf{v}, \mathbf{q}^{(i)}, \mathbf{v}^{(i)}, i < j]$
For $i = 1, \ldots, j-1$ **do**:
    $\alpha = (\mathbf{q}, \mathbf{q}^{(i)})$
    $\mathbf{q} = \mathbf{q} - \alpha\mathbf{q}^{(i)}$
    $\mathbf{v} = \mathbf{v} - \alpha\mathbf{v}^{(i)}$
End for
$\mathbf{q} = \mathbf{q}/\|\mathbf{q}\|_2$
$\mathbf{v} = \mathbf{v}/\|\mathbf{q}\|_2$

$\|.\|_2$ is the Euclidean norm (3.6).

Computational work

The orthonormalization process used in Algorithm 3.2 could make the PGCR algorithm expensive. First of all, the vectors $\mathbf{q}^{(1)}, \ldots, \mathbf{q}^{(j)}$ and $\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(j)}$ need to be stored in memory and with every iteration the memory usage increases. Secondly, the orthonormalization work of the vectors increases with every iteration.
The storage and computing work can be reduced by applying the following techniques:

- Restarting: stop the PGCR algorithm after j_{res} iterations and remove $\mathbf{q}^{(1)}, \ldots, \mathbf{q}^{(j_{res})}$ and $\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(j_{res})}$.
- Truncation: allow only j_{trunc} vectors $\mathbf{q}^{(j)}$ and $\mathbf{v}^{(j)}$, and replace an old vector by a new one.

When restarting or truncation is applied the optimality property of the PGCR algorithm gets lost.

Robustness and convergence

Inspection of the PGCR algorithm in combination with modified Gram-Schmidt orthonormalization shows that breakdown can occur if $\|\mathbf{q}^{(j)}\|_2 = 0$. This can happen if $\mathbf{r}^{(j-1)} = \mathbf{r}^{(j-2)}$, which is unlikely to happen in practice, or if the exact solution is reached, *i.e.* $\mathbf{x}^{(j-1)} = \mathbf{x}$. This means that the PGCR method is very robust.
Concerning convergence of Krylov subspace methods one can show that convergence is monotone,

$$\|\mathbf{r}^{(j)}\| \leq \|\mathbf{r}^{(j-1)}\|. \tag{3.13}$$

3.3 Deflated Krylov subspace methods

The deflation method is originally proposed in Nicolaides [11]. In the late nineties this method had a rebirth as an application for solving elliptic layered

problems with extreme contrasts in the coefficients, see for example Vuik *et al.* [29] or Vuik *et al.* [30]. The matrix of such a system arising from discretization is ill-conditioned because of the large jumps in the coefficients. It was observed that solving the system with a conventional (preconditioned) Krylov subspace method gave erratic convergence behaviour. Removing the smallest eigenvalues of the matrix by the deflation technique solved this problem.
Now the basic idea of deflation will be discussed briefly. More can be found in Frank & Vuik [5]. In Chapter 7 numerical experiments done with deflation (in MATLAB as well as X-stream) are described.

Basic idea of deflation

Consider the linear algebraic $n \times n$ system

$$A\mathbf{x} = \mathbf{b}, \tag{3.14}$$

where A is a general matrix. Let P and Q be given by

$$P = I - AZ(Y^TAZ)^{-1}Y^T,$$
$$Q = I - Z(Y^TAZ)^{-1}Y^TA,$$

where Z and Y are matrices whose columns span suitable subspaces. We have the following properties for P and Q:

- $P^2 = P, \quad Q^2 = Q,$
- $PAZ = Y^TP = 0, \quad Y^TAQ = QZ = 0,$
- $PA = AQ.$

To solve the system $A\mathbf{x} = \mathbf{b}$ using deflation, note that $\mathbf{x}$ can be written as

$$\mathbf{x} = (I - Q)\mathbf{x} + Q\mathbf{x} \tag{3.15}$$

and that $(I - Q)\mathbf{x} = Z(Y^TAZ)^{-1}Y^TA\mathbf{x} = Z(Y^TAZ)^{-1}Y^T\mathbf{b}$ can be computed immediately. Furthermore, $Q\mathbf{x}$ can be obtained solving the deflated system

$$PA\tilde{\mathbf{x}} = P\mathbf{b} \tag{3.16}$$

for $\tilde{\mathbf{x}}$ and premultiplying the result with Q. For a derivation see Appendix A.
Deflation can be combined with preconditioning. Suppose M is a suitable preconditioner, then (3.16) can be replaced by the following: solve $\tilde{\mathbf{x}}$ from

$$M^{-1}PA\tilde{\mathbf{x}} = M^{-1}P\mathbf{b} \tag{3.17}$$

and form $Q\tilde{\mathbf{x}}$. This system can be solved by a Krylov subspace solver.
Note that when the deflated system (3.16) is solved with a Krylov subspace method one always has to take $\tilde{\mathbf{x}}^{(0)} = 0$. Also note that when an initial guess $\mathbf{x}^{(0)} \neq 0$ is chosen, one has to solve the deflated system

$$PA\tilde{\mathbf{v}} = P\mathbf{f}, \tag{3.18}$$

where $\mathbf{v} = \mathbf{x} - \mathbf{x}^{(0)}$ and $\mathbf{f} = \mathbf{b} - A\mathbf{x}^{(0)}$ in order for system (3.14) to be solved.
We will restrict ourselves to the deflated PCG algorithm (DPCG) for the case

that A and M are SPD, and to the deflated PGCR algorithm (DPGCR) for the case that A and M are general matrices. The following algorithm is the deflated variant of Algorithm 3.1.

Algorithm 3.3 (DPCG)
Given an SPD $n \times n$ matrix A, a vector $\mathbf{b}$, an SPD preconditioner M, projectors P and $Q = P^T$ for the case $Y = Z$ and an initial guess $\mathbf{x}^{(0)}$, this algorithm solves the linear system $A\mathbf{x} = \mathbf{b}$.
$\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$
$\tilde{\mathbf{x}}^{(0)} = 0$
$\tilde{\mathbf{r}}^{(0)} = P\mathbf{r}^{(0)}$
$\mathbf{p}^{(0)} = \mathbf{z}^{(0)} = M^{-1}\tilde{\mathbf{r}}^{(0)}$
For $j = 1, \ldots,$ until convergence **do**:
$\alpha = (\tilde{\mathbf{r}}^{(j-1)}, \mathbf{z}^{(j-1)})/(\mathbf{p}^{(j-1)}, PA\mathbf{p}^{(j-1)})$
$\tilde{\mathbf{x}}^{(j)} = \tilde{\mathbf{x}}^{(j-1)} + \alpha\mathbf{p}^{(j-1)}$
$\tilde{\mathbf{r}}^{(j)} = \tilde{\mathbf{r}}^{(j-1)} - \alpha PA\mathbf{p}^{(j-1)}$
$\mathbf{z}^{(j)} = M^{-1}\tilde{\mathbf{r}}^{(j)}$
$\beta = (\tilde{\mathbf{r}}^{(j)}, \mathbf{z}^{(j)})/(\tilde{\mathbf{r}}^{(j-1)}, \mathbf{z}^{(j-1)})$
$\mathbf{p}^{(j)} = \mathbf{z}^{(j)} + \beta\mathbf{p}^{(j-1)}$
End for
$\mathbf{x} \approx (I - Q)A^{-1}\mathbf{r}^{(0)} + Q\tilde{\mathbf{x}}^{(j-1)} + \mathbf{x}^{(0)}$

The deflated variant of Algorithm 3.2 is given by the following algorithm:

Algorithm 3.4 (DPGCR)
Given an $n \times n$ matrix A, a vector $\mathbf{b}$, a preconditioner M, projectors P and Q and an initial guess $\mathbf{x}^{(0)}$, this algorithm solves the linear system $A\mathbf{x} = \mathbf{b}$.
$\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$
$\tilde{\mathbf{x}}^{(0)} = 0$
$\tilde{\mathbf{r}}^{(0)} = P\mathbf{r}^{(0)}$
For $j = 1, \ldots,$ until convergence **do**:
$\mathbf{v}^{(j)} = M^{-1}\tilde{\mathbf{r}}^{(j-1)}$
$\mathbf{q}^{(j)} = PA\mathbf{v}^{(j)}$
$[\mathbf{q}^{(j)}, \mathbf{v}^{(j)}] = \mathbf{orthonorm}[\mathbf{q}^{(j)}, \mathbf{v}^{(j)}, \mathbf{q}^{(i)}, \mathbf{v}^{(i)}, i < j]$
$\beta = (\tilde{\mathbf{r}}^{(j-1)}, \mathbf{q}^{(j)})$
$\tilde{\mathbf{x}}^{(j)} = \tilde{\mathbf{x}}^{(j-1)} + \beta\mathbf{v}^{(j)}$
$\tilde{\mathbf{r}}^{(j)} = \tilde{\mathbf{r}}^{(j-1)} - \beta\mathbf{q}^{(j)}$
End for
$\mathbf{x} \approx (I - Q)A^{-1}\mathbf{r}^{(0)} + Q\tilde{\mathbf{x}}^{(j-1)} + \mathbf{x}^{(0)}$

Choosing Z and Y

There are various possibilities to choose the matrices Z and Y. In general Y is often chosen equal to Z. The columns of Z are usually called *deflation vectors*. For example, approximate eigenvectors can be chosen. Details can be found in Vuik *et al.* [29] and Vuik *et al.* [30]. In Chapter 7 constant deflation (CD) and constant linear deflation (CLD) are described.

3.4 Solving the stationary incompressible Navier-Stokes equations

There are various methods to solve the Navier-Stokes equations iteratively. A popular method used in engineering is the Semi-Implicit Method for Pressure-Linked Equations (SIMPLE). Many improved variants of SIMPLE have been proposed, like for example the SIMPLE Revised (SIMPLER) method. However, in engineering literature, for example Patankar [13] or Ferziger & Perić [3], it is not easy to verify which algebraic systems are actually solved because the presented algorithms are overrun with details. Therefore, a more mathematically convenient way is to present them in a so-called distributive iterative framework.
In this section we will focus on SIMPLE. Distributive iteration will be described and we will show that SIMPLE can be written as a classical distributive iterative method.

The algebraic system to be solved

We consider a colocated grid. If we delete the time derivative in equation (2.23) and linearize $N(\mathbf{u})$, the algebraic system to be solved may be denoted as

$$\begin{pmatrix} N & G \\ D & C \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix}. \tag{3.19}$$

Distributive iteration

Suppose we have a linear system

$$A\mathbf{x} = \mathbf{b}. \tag{3.20}$$

The system is postconditioned:

$$AB\hat{\mathbf{x}} = \mathbf{b}, \quad \mathbf{x} = B\hat{\mathbf{x}}, \tag{3.21}$$

where B is the postconditioning matrix, chosen in such a way that (3.21) is easier to solve iteratively than (3.20). For example, one can choose AB to be an M-matrix (see Wesseling [32]), while A is not. Splitting the matrix AB yields

$$AB = M - N,$$

corresponding to the following splitting of the original matrix A:

$$A = MB^{-1} - NB^{-1}.$$

This leads to the following stationary iterative method for (3.20):

$$MB^{-1}\mathbf{x}^{(k+1)} = NB^{-1}\mathbf{x}^{(k)} + \mathbf{b},$$

or

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + BM^{-1}(\mathbf{b} - A\mathbf{x}^{(k)}). \tag{3.22}$$

Note that the iterative method defined by (3.22) is consistent for any non-singular B and M: if it converges, it solves $A\mathbf{x} = \mathbf{b}$. The method is called

distributive iteration because the correction $M^{-1}(\mathbf{b} - A\mathbf{x}^{(k)})$ is distributed, so to speak, over the elements of $\mathbf{x}^{(k+1)}$.

Distributive iteration for the stationary Navier-Stokes equations

The system to be solved is (3.19). Define

$$A = \begin{pmatrix} N & G \\ D & C \end{pmatrix}.$$

A distribution matrix B such that AB is of block-triangular form:

$$AB = \begin{pmatrix} Q & 0 \\ R & S \end{pmatrix}.$$

would be attractive. A possible choice for B is

$$B = \begin{pmatrix} I & B_{12} \\ 0 & B_{22} \end{pmatrix},$$

resulting in

$$AB = \begin{pmatrix} N & NB_{12} + GB_{22} \\ D & DB_{12} + CB_{22} \end{pmatrix}.$$

Choosing B such that $NB_{12} + GB_{22} = 0$ gives

$$B_{12} = -N^{-1}GB_{22},$$

resulting in

$$AB = \begin{pmatrix} N & 0 \\ D & E \end{pmatrix}, \tag{3.23}$$

with $E = (C - DN^{-1}G)B_{22}$.

SIMPLE method

The SIMPLE method is obtained by choosing $B_{22} = I$, so that (3.23) becomes

$$AB = \begin{pmatrix} N & 0 \\ D & C - DN^{-1}G \end{pmatrix}.$$

A splitting $AB = M - N$ is defined by

$$M = \begin{pmatrix} Q & 0 \\ D & R \end{pmatrix},$$

where Q and R are approximations to N and $C - DN^{-1}G$ such that $M\mathbf{x} = \mathbf{b}$ is easily solvable. For the distribution step in (3.22) B is approximated by

$$B = \begin{pmatrix} I & -\tilde{N}^{-1}G \\ 0 & I \end{pmatrix},$$

where $\tilde{N}^{-1}$ is an easy to evaluate approximate inverse of N. In the original SIMPLE method, one chooses $\tilde{N} = diag(N)$; this makes $D\tilde{N}^{-1}G$ easy to determine.

Now the SIMPLE method is given by (3.22). In the present case we have

$$\mathbf{b} - A\mathbf{x}^{(k)} = \begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix} - \begin{pmatrix} N & G \\ D & C \end{pmatrix} \begin{pmatrix} \mathbf{u}^{(k)} \\ \mathbf{p}^{(k)} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_1^{(k)} \\ \mathbf{r}_2^{(k)} \end{pmatrix}. \tag{3.24}$$

Algorithm 3.5 (SIMPLE)
Let $\mathbf{p}^{(0)}$ and $\mathbf{u}^{(0)}$ be initial estimations of the pressure and velocity field, respectively. Let ω_p and ω_u be underrelaxation parameters. This algorithm solves the non-linear system (3.19) for the stationary incompressible Navier-Stokes equations.
For $k = 0, \ldots,$ until convergence **do**:
Compute $\mathbf{r}_1^{(k)}$ and $\mathbf{r}_2^{(k)}$ from (3.24)
Solve $Q\delta\mathbf{u} = \mathbf{r}_1^{(k)}$
Solve $R\delta\mathbf{p} = \mathbf{r}_2^{(k)} - D\delta\mathbf{u}$
$\delta\mathbf{u} = \delta\mathbf{u} - \tilde{N}^{-1}G\delta\mathbf{p}$
$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \omega_u\delta\mathbf{u}$
$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} + \omega_p\delta\mathbf{p}$
End for
$\mathbf{u} \approx \mathbf{u}^{(k)}$
$\mathbf{p} \approx \mathbf{p}^{(k)}$

Experience has shown that the use of underrelaxation factors is necessary. Many variants of the SIMPLE method can also be described in the presented distributed iteration framework, see for example Vuik *et al.* [28] or Vuik & Saghir [27] for a description of the SIMPLER method.

Convergence, computing work and stopcriterion

One can show that SIMPLE converges slowly and that the computing work is of $\mathcal{O}(\mathcal{N}^2)$, just like BIMs. Fortunately, like BIMs, the SIMPLE method is suitable for acceleration by Krylov subspace methods or multigrid.
Besides making the residuals small it is recommendable to make the velocity field also sufficiently divergence free, *i.e.* to make $D\mathbf{u}^{(k)}$ sufficiently small, for instance

$$\|D\mathbf{u}^{(k)}\|_\infty < \varepsilon V/H, \quad 0 < \varepsilon \leq 1,$$

with $\|\mathbf{x}\|_\infty = max\{|x_1|, \ldots, |x_n|\}$, V and H typical magnitudes for the velocity and domain, and ε a certain fixed tolerance. See for more details Wesseling [32].

4. DOMAIN DECOMPOSITION METHODS

The term domain decomposition (DD) has slightly different meanings to specialists within the discipline of PDEs. In parallel computing it means decomposing data from a computational model among the processors in a distributed memory computer. In asymptotic analysis it means separation of the physical domain into regions that can be modeled with different equations. In preconditioning methods, DD refers to the process of subdividing the solution of a large linear system into smaller problems whose solutions can be used to produce a preconditioner (or solver) for the system of equations that results from discretizing the PDE on the entire domain. Note that all three of these interpretations may occur in a single program.
In this chapter we will concentrate on DD methods as iterative solution methods for solving PDEs based on a decomposition of the spatial domain of the problem into several subdomains.
There are several motivations for using DD:

- ease of parallelization and good parallel performance,
- simplification of problems on a complicated geometry,
- good convergence properties,
- different physical models can be used in different subdomains,
- local grid refinement can be implemented with more ease,
- memory requirements can be reduced, because the subproblems can be much smaller than the total problem.

The structure of this chapter is as follows. In Section 4.1 basic DD methods are discussed, in Section 4.2 DD for the stationary incompressible Navier-Stokes equations is treated.

4.1 Basic domain decomposition methods

A survey on DD methods can be found in for example Smith *et al.* [15], Saad [14] or Brakkee [1]. The most common DD methods are Schwarz methods. These methods are used mostly for overlapping domains. The Schwarz method for non-overlapping domains does not converge, but can be used as a preconditioner. For non-overlapping domains that have a joint interface, Schur complement methods are used mostly. Non-overlapping methods differ from overlapping methods by solving an additional interface equation. In this thesis only Schwarz methods for overlapping domains are described. Schur

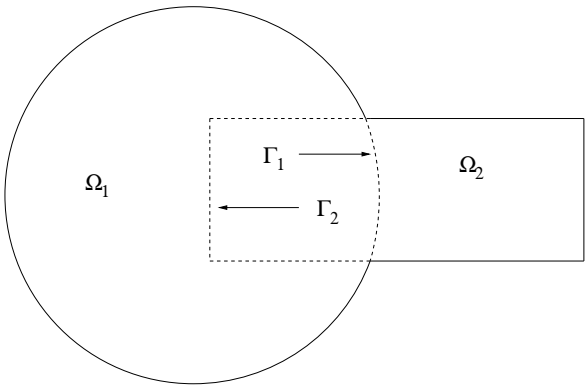

Fig. 4.1: An overlapping decomposition of the domain

complement methods are treated in, for example, Brakkee [1].

DD aims to solve the differential equation

$$\mathcal{L}u = f \quad \text{on } \Omega, \tag{4.1}$$

with suitable BCs on $\partial\Omega$, by decomposing the domain into subdomains $\bar{\Omega} = \bar{\Omega}_1 \cup \ldots \cup \bar{\Omega}_k$, where Ω is open and the Ω_i are open subsets of Ω, as illustrated in Figure 4.1. For simplicity, we consider two subdomains $\bar{\Omega} = \bar{\Omega}_1 \cup \bar{\Omega}_2$ and we apply the Dirichlet BC $u = g$ on $\partial\Omega$, although more general Neumann and Robin BCs may also be dealt with.

In this section we restrict ourselves to so-called one-level DD methods. In Subsection 4.1.1 the basic Schwarz method is discussed, followed by the multiplicative Schwarz method in Subsection 4.1.2 and the additive Schwarz method in Subsection 4.1.3. In Subsection 4.1.4 convergence properties of the Schwarz method are treated, without going into details.

4.1.1 Basic Schwarz method

The simplest and earliest known DD method is the one-level Schwarz method. We will start this subsection by describing the basic Schwarz method and after that make a distinction between the multiplicative and additive Schwarz method.

Consider the overlapping domain, as shown in Figure 4.1, with $\Omega = \Omega_1 \cup \Omega_2$.
The part of the boundary of Ω_i which is located in the interior of Ω_j $(j \neq i)$ is denoted by Γ_i.
The basic Schwarz method to solve (4.1) begins by selecting an initial guess $u_2^{(0)}$, for the values in Ω_2. Then, iteratively for $k = 1, 2, \ldots$, one solves the boundary

value problem,

$$\begin{cases} \mathcal{L}u_1^{(k)} = f & \text{in } \Omega_1, \\ u_1^{(k)} = g & \text{on } \partial\Omega_1 \setminus \Gamma_1, \\ u_1^{(k)} = u_2^{(k-1)}|_{\Gamma_1} & \text{on } \Gamma_1, \end{cases}$$

for $u_1^{(k)}$. This is followed by the solution of the boundary value problem,

$$\begin{cases} \mathcal{L}u_2^{(k)} = f & \text{in } \Omega_2, \\ u_2^{(k)} = g & \text{on } \partial\Omega_2 \setminus \Gamma_2, \\ u_2^{(k)} = u_1^{(k)}|_{\Gamma_2} & \text{on } \Gamma_2. \end{cases}$$

The k-th iterate is then defined as

$$u^{(k)}(x,y) = \begin{cases} u_1^{(k)}(x,y) & \text{if } (x,y) \in \Omega \setminus \Omega_2, \\ u_2^{(k)}(x,y) & \text{if } (x,y) \in \Omega_2. \end{cases}$$

It is shown (see Brakkee [1] for some references) that for self-adjoint elliptic operators $\mathcal{L}$, the iterates converge in the norm induced by the operator to the true solution u of (4.1) like

$$\|u - u^{(k)}\|_{\mathcal{L}} \leq \rho^k \|u - u^{(0)}\|_{\mathcal{L}}.$$

Here, ρ depends on the choice of Ω_1 and Ω_2.

The discrete form of (4.1) is denoted by

$$A\mathbf{y} = \mathbf{b}, \tag{4.2}$$

where A represents the discretization of $\mathcal{L}$ and BCs on the global domain. We restrict ourselves to the case that the grids of the subdomains coincide in the overlap area.

The algebraic Schwarz algorithm will be described in matrix notation. Let I_1 and I_2 be the index sets of unknowns in the interior of Ω_1 and Ω_2, respectively. The total number of unknowns is $n = |I|$ and n_i denotes the number of unknowns in subdomain Ω_i. For the case of generous overlap $I_1 \cap I_2 \neq \emptyset$.
Denote by R_i^T a trivial extension matrix of dimension $n \times n_i$, defined as

$$(R_i^T y_i)_k = \begin{cases} (y_i)_k & \text{if } k \in I_i \\ 0 & \text{else,} \end{cases}$$

for $y_i \in \mathbb{R}^{n_i}$. The entries in the matrix R_i^T consist of ones and zeroes with at most one '1' in each row. The transpose R_i is a trivial restriction matrix whose action is to restrict a full length vector of size n to a subdomain vector of size n_i, by selecting the components of the vector corresponding to I_i. Note that $R_i R_i^T = I_{n_i}$, while $R_i^T R_i \neq I_n$. The local subdomain matrices are written in terms of the global matrix A and the restriction matrices R_i as

$$A_{11} = R_1 A R_1^T, \quad A_{22} = R_2 A R_2^T.$$

4.1.2 Multiplicative Schwarz method

The algebraic Schwarz iteration starts with an arbitrary initial guess $\mathbf{v}^{(0)}$ and constructs a sequence of approximations as follows:

$$\mathbf{y}^{(k+1/2)} = \mathbf{y}^{(k)} + R_1^T A_{11}^{-1} R_1(\mathbf{b} - A\mathbf{y}^{(k)}), \tag{4.3}$$

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k+1/2)} + R_2^T A_{22}^{-1} R_2(\mathbf{b} - A\mathbf{y}^{(k+1/2)}). \tag{4.4}$$

Note that the R_i matrices are never formed in practice. When $I_1 \cap I_2 = \emptyset$ this is an instance of block Gauss-Seidel iteration. When $I_1 \cap I_2 \neq \emptyset$, (4.3) and (4.4) form a generalization of block Gauss-Seidel for overlapping subblocks of the matrix A.

Introducing the error $\epsilon^{(i)} = \mathbf{y} - \mathbf{y}^{(i)}$, (4.3) and (4.4) are rewritten as

$$\epsilon^{(k+1/2)} = (I - P_1)\epsilon^{(k)}, \tag{4.5}$$

$$\epsilon^{(k+1)} = (I - P_2)\epsilon^{(k+1/2)}, \tag{4.6}$$

with $P_i = R_i^T A_{ii}^{-1} R_i A$. The matrices P_i present projection operators ($P_i^2 = P_i$). Combining (4.5) and (4.6), we get

$$\epsilon^{(k+1)} = (I - P_2)(I - P_1)\epsilon^{(k)},$$

which is the reason for calling the algorithm *multiplicative*. We may write (4.3) and (4.4) also as a preconditioned Richardson iteration

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + M_{gs}^{-1}(\mathbf{b} - A\mathbf{y}^{(k)}),$$

with $M_{gs}^{-1}A = I - (I - P_2)(I - P_1)$. For the case of disjoint index sets $I_1 \cap I_2 = \emptyset$, M_{gs} is the block lower triangular matrix of A. Note that the multiplicative Schwarz method is not suitable for parallelization, because all blocks are handled consecutively, one after the other (see (4.3) and (4.4)).

4.1.3 Additive Schwarz method

A block Jacobi variant of (4.3) and (4.4) is

$$\mathbf{y}^{(k+1/2)} = \mathbf{y}^{(k)} + R_1^T A_{11}^{-1} R_1(\mathbf{b} - A\mathbf{y}^{(k)}), \tag{4.7}$$

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k+1/2)} + R_2^T A_{22}^{-1} R_2(\mathbf{b} - A\mathbf{y}^{(k)}). \tag{4.8}$$

Its main advantage is that the subdomain corrections can be carried out in parallel, in contrast with equations (4.3) and (4.4). Writing (4.7) and (4.8) in terms of the error $\epsilon^{(i)}$, we get

$$\epsilon^{(k+1)} = (I - P_1 - P_2)\epsilon^{(k)}.$$

Equations (4.7)-(4.8) may also be written as a preconditioned Richardson iteration

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + M_{jac}^{-1}(\mathbf{b} - A\mathbf{y}^{(k)}),$$

with $M_{jac}^{-1}A = P_1 + P_2$, which is the reason for calling this algorithm *additive*. Note that the additive Schwarz method is more suitable for parallelization than the multiplicative Schwarz algorithm. However, concerning convergence it can be observed that the multiplicative Schwarz method converges faster than the additive Schwarz method.
In general, DD methods converge slowly, so some acceleration technique is required. Several of those techniques are found in the literature: multigrid acceleration and acceleration by Krylov subspace methods. For references see Brakkee [1].

4.1.4 Convergence properties

Because DD methods converge slowly in general, some acceleration technique is required. For multigrid acceleration the reader is referred to Brakkee [1] where some useful references can be found. In this subsection acceleration by Krylov subspace methods will be described briefly.

Krylov subspace methods are frequently used to accelerate DD methods but mostly in conjunction with an approximate solution of the subdomains. Numerical experiments show that Krylov subspace acceleration in general provides significant acceleration of convergence, and that it makes convergence more robust with respect to the number of subdomains, mesh size and external BCs. A DD method accelerated by a Krylov subspace method is called a *Krylov-Schwarz* method. Equivalently, Schwarz-Krylov means that DD is used as a preconditioner for a Krylov subspace method.
Krylov subspace acceleration amounts to the solution of a preconditioned system of equations

$$M^{-1}A\mathbf{y} = M^{-1}\mathbf{b}, \tag{4.9}$$

with $M = M_{gs}$ or $M = M_{jac}$. In case of a domain with k subdomains, $M_{gs}^{-1}A = I - (I - P_k)(I - P_{k-1})\dots(I - P_1)$ and $M_{jac}^{-1}A = \sum_{i=1,\dots,k} P_i$. Convergence theory is most developed for symmetric problems (with A an SPD matrix), using acceleration by conjugate gradients. For this purpose, $M^{-1}A$ must be symmetric.
Assume that $M^{-1}A$ is SPD, then the error after m conjugate gradient accelerations $\mathbf{e}^{(m)} = \mathbf{v} - \mathbf{v}^{(m)}$, behaves like

$$\|\mathbf{e}^{(m)}\|_{M^{-1}A} \leq 2c^m \|\mathbf{e}^{(0)}\|_{M^{-1}A}, \quad c = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}, \tag{4.10}$$

where

$$\kappa = \frac{\lambda_{max}(M^{-1}A)}{\lambda_{min}(M^{-1}A)}.$$

Formula (4.10) is an upper bound: the rate of convergence depends not only on the extremal eigenvalues, but on the whole spectrum. For instance, with clustering of eigenvalues, convergence will typically be faster. In general, however, convergence of iterative methods improves when the condition number of the preconditioned matrix decreases.

The mesh size and number of subdomains have influence on the condition number. Only the results will be given, for the derivation the reader is referred to Brakkee [1]. Let the mesh size be given by h. For the Schwarz algorithm with minimal overlap,

$$\kappa(M_{jac}^{-1}A) = \mathcal{O}(h^{-1}H^{-1}),$$

and with generous overlap,

$$\kappa(M_{jac}^{-1}A) = \mathcal{O}(1/H^2).$$

For the effect of the number of subdomains on the convergence, we consider local coupling and global coupling. *Local coupling* means that each subdomain only interacts through coupling with its neighbors. In this case the condition number grows with the number of subdomains. With *global coupling* all subdomains communicate with each other. In this case the dependence of the condition number on the number of subdomains can be removed by using a so-called *coarse grid correction*. Two types of coarse grid correction methods can be distinguished: two-grid coarse grid correction and deflation coarse grid correction.
Two-grid coarse grid correction results in a two-level algorithm. The basic idea is to construct a preconditioner that reduces both low and high frequency error components. For the additive Schwarz method the block Jacobi preconditioner is used as a smoother, see for more details Smith [15]. Other references are Nabben [10] and Padiy *et al.* [12].
Another coarse grid correction approach is to use a *deflation coarse grid correction*, as described in Section 3.3. It turns out that deflation in combination with DD gives better results compared to two-grid coarse grid correction.

Deflation and domain decomposition

Various publications can be found on deflation in combination with DD and parallel computing. Some useful references are Frank & Vuik [5], Vuik & Frank [25], Vuik & Frank [24], Vermolen & Vuik [20] and Keyes [8].
Choosing the matrices Z and Y, as described in Section 3.3, is usually done at subdomain level. This choice is further investigated in Chapter 7.1.

4.2 Domain decomposition for the incompressible Navier-Stokes equations

In this section the additive Schwarz DD method is combined with the SIMPLE method for solving the incompressible Navier-Stokes equations. The system to be solved is given by (3.19). For the two-dimensional Navier-Stokes equations ($d = 2$) the system becomes

$$A = \begin{bmatrix} N_1 & 0 & G_1 \\ 0 & N_2 & G_2 \\ D_1 & D_2 & C \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{p} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix}. \tag{4.11}$$

For reasons given in the previous sections, we accelerate the additive Schwarz method with the PGCR method or with the DPGCR method when a deflation

coarse grid correction is necessary. The resulting methods are referred to as the PGCR-Schwarz method and the DPGCR-Schwarz method, respectively. The SIMPLE method can be accelerated by the PGCR method, resulting in the PGCR-SIMPLE method.
First we will have to choose whether we apply DD in SIMPLE, or SIMPLE in DD. Only literature could be found on DD in SIMPLE (see for example Teigland [16]) so we will continue with this.

First PGCR-SIMPLE will be discussed. This method can be found in Li & Vuik [9]. The preconditioner M in PGCR is chosen as BP^{-1} defined by (see Subsection 3.4)

$$B = \begin{bmatrix} I & 0 & -\tilde{N}_1^{-1}G_1 \\ 0 & I & -\tilde{N}_2^{-1}G_2 \\ 0 & 0 & I \end{bmatrix} \tag{4.12}$$

and

$$P = \begin{bmatrix} N_1 & 0 & 0 \\ 0 & N_2 & 0 \\ D_1 & D_2 & C - \sum_{i=1}^{2} D_i \tilde{N}_i^{-1} G_i \end{bmatrix}, \tag{4.13}$$

with $\tilde{N}_i = diag(N_i)$.
Within each PGCR-SIMPLE iteration $j = 1, \ldots, n_1$ the search directions $\mathbf{v}^{(j)} = BP^{-1}\mathbf{r}^{(j-1)}$ have to be computed. The vector $\mathbf{v}^{(j)}$ can be computed by applying the following distributive step:

1. Solve $P\mathbf{w} = \mathbf{r}^{(j-1)}$;
2. $\mathbf{v}^{(j)} = B\mathbf{w}$;

where w is an auxiliary variable.
Substitution of B (4.12) and P (4.13) yields

1. (a) Solve $N_i w_i = r_i^{j-1}, \quad i = 1, \ldots, d$;
   (b) Solve $(C - \sum_{i=1}^{d} D_i \tilde{N}_i^{-1} G_i) w_{d+1} = r_{d+1}^{j-1} - \sum_{i=1}^{d} D_i w_i$;
2. $\mathbf{v}^{(j)} = B\mathbf{w}$;

The rest of the steps in PGCR-SIMPLE is the same as in the PGCR method (Algorithm 3.2).

Now DD is applied to the systems in step 1 and 2: for the d systems in step 1 the PGCR algorithm is used with the block Jacobi preconditioner blockdiag(N_i), for the system in step 2 the DPGCR algorithm is used with the block Jacobi preconditioner blockdiag($C - \sum_{i=1}^{d} D_i \tilde{N}_i^{-1} G_i$). This is done because the matrix in the system of step 2 can be seen as a discrete Laplacian operator.
Writing out the distributive steps to avoid computations of inverses leads to the following algorithm. Note that deflation is only written out till a certain level; systems involving the matrix Z are not explicitly written out.

Algorithm 4.1 (DPGCR-Schwarz in PGCR-SIMPLE)
Consider the system $A\mathbf{y} = \mathbf{b}$ resulting from the discretization of the Navier-Stokes equations. Let $\mathbf{y}^{(0)}$ be an initial guess of the system, $Y = Z$ and Z a certain subspace. Then this algorithm describes how the deflated PGCR Krylov subspace accelerated additive Schwarz DD method is applied within the PGCR accelerated SIMPLE. $\tilde{\mathbf{w}}$ is an approximation of $\mathbf{w}$ and α is a relaxation parameter.

$\mathbf{r}_i^{(0)} = \mathbf{b}_i - N_i\mathbf{y}_i^{(0)} - G_i\mathbf{y}_{d+1}^{(0)}, \quad i = 1, \ldots, d;$
$\mathbf{r}_{d+1}^{(0)} = \mathbf{b}_{d+1} - \sum_{i=1}^{d} D_i\mathbf{y}_i^{(0)} - C\mathbf{y}_{d+1}^{(0)};$
For $j = 1, \ldots, n_1$ **do**:
$\mathbf{w}_i = \mathbf{pgcrschw}[N_i, \mathbf{r}_i^{(j-1)}, \tilde{\mathbf{w}}_i], \quad i = 1, \ldots, d;$
$\mathbf{w}_{d+1} = \mathbf{dpgcrschw}[C - \sum_{i=1}^{d} D_i\tilde{N}_i^{-1}G_i, \mathbf{r}_{d+1}^{(j-1)} - \sum_{i=1}^{d} D_i\mathbf{w}_i, \tilde{\mathbf{w}}_{d+1}];$
$\mathbf{v}_i^{(j)} = \alpha_i(\mathbf{w}_i - \tilde{N}_i^{-1}G_i\mathbf{w}_i), \quad i = 1, \ldots, d;$
$\mathbf{v}_{d+1}^{(j)} = \alpha_{d+1}\mathbf{w}_{d+1};$
$\mathbf{q}_i^{(j)} = N_i\mathbf{v}_i^{(j)} + G_i\mathbf{v}_{d+1}^{(j)}, \quad i = 1, \ldots, d;$
$\mathbf{q}_{d+1}^{(j)} = \sum_{i=1}^{d} D_i\mathbf{v}_i^{(j)} + C\mathbf{v}_{d+1}^{(j)};$
$[\mathbf{q}^{(j)}, \mathbf{v}^{(j)}] = \mathbf{orthonorm1}[\mathbf{q}^{(j)}, \mathbf{v}^{(j)}, \mathbf{q}^{(i)}, \mathbf{v}^{(i)}, i < j];$
$\gamma = (\mathbf{r}^{(j-1)}, \mathbf{q}^{(j)});$
$\mathbf{y}^{(j)} = \mathbf{y}^{(j-1)} + \gamma\mathbf{v}^{(j)};$
$\mathbf{r}^{(j)} = \mathbf{r}^{(j-1)} - \gamma\mathbf{q}^{(j)};$
End for
$\mathbf{y} \approx \mathbf{y}^{(j)};$

Function $\bar{\mathbf{y}} = \mathbf{pgcrschw}[\bar{A}, \bar{\mathbf{b}}, \bar{\mathbf{y}}^{(0)}]$
$\bar{\mathbf{r}}^0 = \bar{\mathbf{b}} - \bar{A}\bar{\mathbf{y}}^{(0)};$
For $j = 1, \ldots, n_2$ **do**:
Solve $\bar{A}_{mm}\bar{\mathbf{v}}_m^{(j)} = \bar{\mathbf{r}}_m^{(j-1)}, \quad j = 1, \ldots, nblock;$
$\bar{\mathbf{q}}^{(j)} = \bar{A}\bar{\mathbf{v}}^{(j)};$
$[\bar{\mathbf{q}}^{(j)}, \bar{\mathbf{v}}^{(j)}] = \mathbf{orthonorm2}[\bar{\mathbf{q}}^{(j)}, \bar{\mathbf{v}}^{(j)}, \bar{\mathbf{q}}^{(i)}, \bar{\mathbf{v}}^{(i)}, i < j];$
$\bar{\gamma} = (\bar{\mathbf{r}}^{(j-1)}, \bar{\mathbf{q}}^{(j)});$
$\bar{\mathbf{y}}^{(j)} = \bar{\mathbf{y}}^{(j-1)} + \bar{\gamma}\bar{\mathbf{v}}^{(j)};$
$\bar{\mathbf{r}}^{(j)} = \bar{\mathbf{r}}^{(j-1)} - \bar{\gamma}\bar{\mathbf{q}}^{(j)};$
End for
$\bar{\mathbf{y}} \approx \bar{\mathbf{y}}^{(j)};$

Function $\bar{\mathbf{y}} = \mathbf{dpgcrschw}[\bar{A}, \bar{\mathbf{b}}, \bar{\mathbf{y}}^{(0)}]$
$\bar{\mathbf{r}}^{(0)} = \bar{\mathbf{b}} - \bar{A}\bar{\mathbf{y}}^{(0)};$
$\tilde{\mathbf{y}}^{(0)} = 0;$
Solve $Z^T\bar{A}Z\bar{\mathbf{w}}_1 = Z^T\bar{\mathbf{r}}^{(0)};$
$\tilde{\mathbf{r}}^{(0)} = \bar{\mathbf{r}}^{(0)} - \bar{A}Z\bar{\mathbf{w}}_1;$
For $j = 1, \ldots, n_2$ **do**:
Solve $\bar{A}_{mm}\bar{\mathbf{s}}_m^{(j)} = \tilde{\mathbf{r}}^{(j-1)}, \quad j = 1, \ldots, nblock;$
Solve $Z^T\bar{A}Z\bar{\mathbf{w}}_2 = Z^T\bar{A}\bar{\mathbf{s}}^{(j)};$
$\bar{\mathbf{v}}^{(j)} = \bar{A}\bar{\mathbf{s}}^{(j)} - \bar{A}Z\bar{\mathbf{w}}_2;$
$[\bar{\mathbf{v}}^{(j)}, \bar{\mathbf{s}}^{(j)}] = \mathbf{orthonorm3}[\bar{\mathbf{v}}^{(j)}, \bar{\mathbf{s}}^{(j)}, \bar{\mathbf{v}}^{(i)}, \bar{\mathbf{s}}^{(i)}, i < j];$
$\bar{\beta} = (\tilde{\mathbf{r}}^{(j-1)}, \bar{\mathbf{v}}^{(j)});$
$\tilde{\mathbf{y}}^{(j)} = \tilde{\mathbf{y}}^{(j-1)} + \bar{\beta}\bar{\mathbf{s}}^{(j)};$

$\tilde{\mathbf{r}}^{(j)} = \tilde{\mathbf{r}}^{(j-1)} - \bar{\beta}\bar{\mathbf{v}}^{(j)}$;
End for
Solve $Z^T \bar{A} Z \bar{\mathbf{w}}_3 = Z^T \bar{A} \tilde{\mathbf{y}}^{(j-1)}$;
$\bar{\mathbf{y}} \approx Z(\bar{\mathbf{w}}_1 - \bar{\mathbf{w}}_3) + \tilde{\mathbf{y}}^{(j-1)} + \bar{\mathbf{y}}^{(0)}$;

In the above algorithm the routines **dpcgrschw** and **pgcrschw** are called when PGCR-Schwarz DD is applied with or without deflation. In the third argument of this routine an initial guess is provided. The routines **orthonorm1**, **orthonorm2** and **orthonorm3** refer to the orthonormalization process used, for example the modified Gram-Schmidt algorithm, see Section 3.2.
The relaxation parameters $\alpha_i, i = 1, \ldots, d$ in the PGCR-SIMPLE loop refer to the velocity components, the relaxation parameter α_{d+1} to the pressure. When applying a Krylov subspace acceleration it is expected that less relaxation is necessary, so it is likely that relaxation parameters closer to 1 can be chosen.
Finally, the solution procedure for solving the incompressible Navier-Stokes equations is given by the following algorithm:

Algorithm 4.2 (solving the incompressible Navier-Stokes equations)
Let $\mathbf{y}^{(0)}$ be an initial guess. Then this algorithm solves the non-linear system (3.19) for the stationary incompressible Navier-Stokes equations.
For $k = 1, \ldots,$ until convergence **do**
Solve $A(\mathbf{y}^{(k-1)})\mathbf{y}^{(k)} = \mathbf{b}$ with DPGCR-Schwarz in PGCR-SIMPLE
End for
$\mathbf{y} \approx \mathbf{y}^{(k)}$

5. X-STREAM

In this chapter the connection between X-stream and the subjects treated in the previous chapters will be described.

Mathematical model

In X-stream the basic conservation equations as discussed in Section 2.1 can be solved both in the glass melting space as in the combustion space. Some other mathematical models describing physical processes available in X-stream are: combustion, turbulence, radiation, batch, electrical boosting, foam, bubbling and stirring. Most of these models are specifically related to the process of glass melting. Details on this kind of models can be found in the GTM-X user manual [17], Twerda [18] and Verweij [21].

Finite volume discretization

The FV method is used on a colocated grid in the X-stream code. The grid is block-structured and boundary-fitted, see for details Ferziger & Perić [3] or Wesseling [32]. Defect correction (a method to improve the accuracy of a lower order discretization, without having to solve for a higher order discretization, see Wesseling [32]) is applied and blending can be done with several schemes: the UDS scheme, the CDS scheme and higher order schemes. Both the instationary and the stationary (non-dimensionless) incompressible Navier-Stokes equations can be solved. The PWI method is used to resolve the checkerboard problem.

Iterative solution methods

In X-stream, the incompressible Navier-Stokes equations are solved with the SIMPLE method, according to Algorithm 3.5. The linear systems resulting from the SIMPLE method can be solved with the CG method, the SIP method and the Space Tri-Diagonal Matrix Algorithm (SPTDMA) (see Patankar [13]). All these iterative methods are used within a domain decomposition context. Note that the CG solver can only be applied to SPD matrices. In general, the SIP solver is the most robust and efficient solver for small grids and a small number of blocks.

Domain decomposition

In the X-stream code, a Schwarz DD method with the following properties is implemented:

- Unaccelerated additive one-level Schwarz method with minimal overlap;
- Inaccurate subdomain solution by a single iteration of SIP, SPTDMA or CG;
- Block-structured cell-centered grid;
- Local grid refinement can be done at block level;
- Application of different models on different blocks;
- Parallelized using the MPI (Message Passing Interface) standard.

For more details and implementation description, see Verweij [21].

In X-stream, a set of subdomains (blocks) for which the same model is applied is referred to as 'domain'. For the solution to the Navier-Stokes equations a Schwarz iteration is referred to as an *inner iteration* and a SIMPLE iteration as an *outer iteration*.

A global flowchart of the current solution procedure for solving the Navier-Stokes equations in X-stream is given by Figure 5.1. The abbreviations LC and GC denote respectively local and global communication between the processors in a parallel computing environment.
The SIMPLE stabilization iteration (SSI) refers to a method for improving convergence of the pressure-correction system. Each SSI iteration can be seen as a SIMPLE step without solving the pseudo velocities.

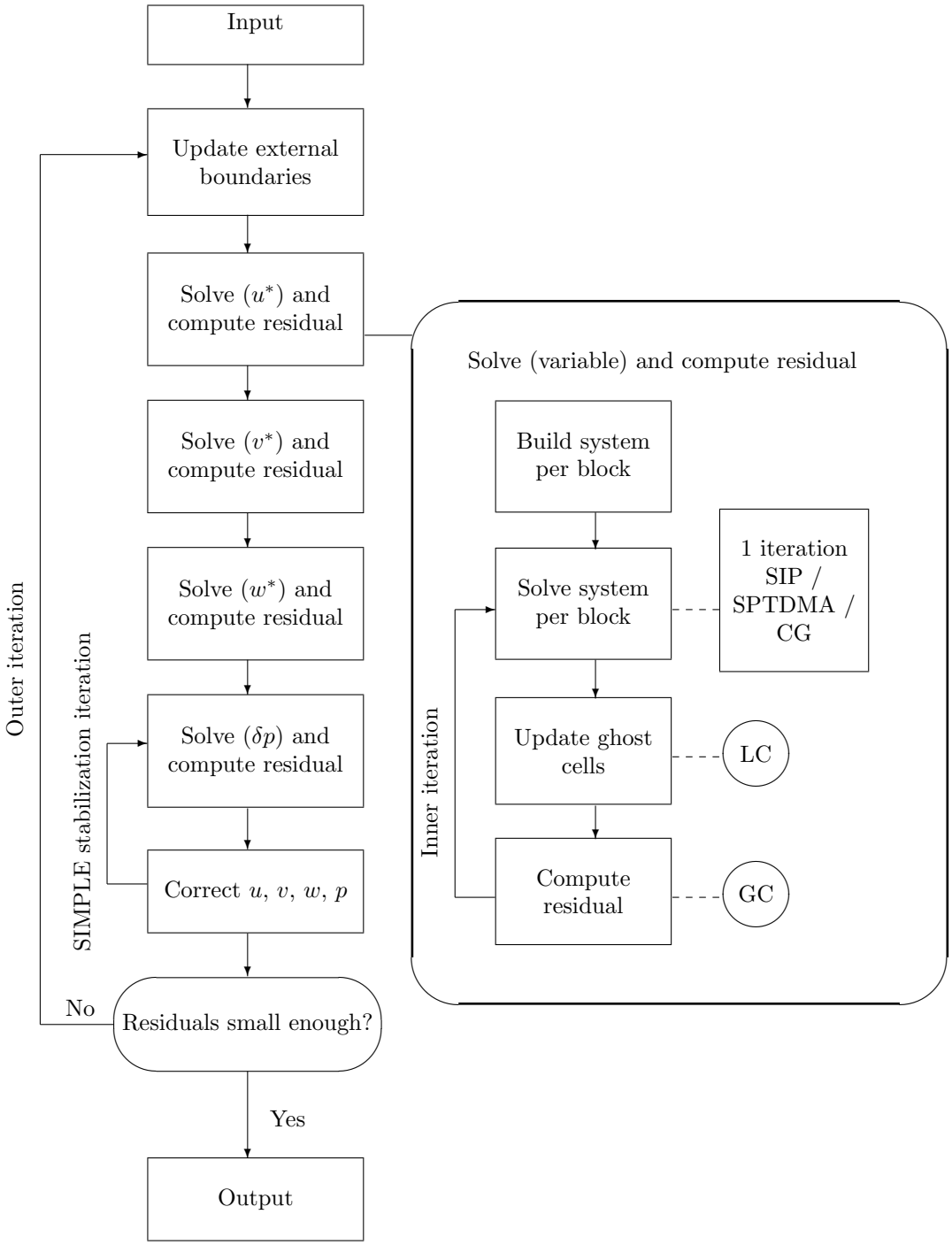

Fig. 5.1: Flowchart of the solution procedure in X-stream for solving the stationary incompressible Navier-Stokes equations

6. TEST CASES IN X-STREAM

In this chapter four test cases in X-stream are described. These test cases are used for experiments in Chapter 7 and Chapter 8. Varying from relatively simple to complex, the test cases we consider are: a pipe flow, a flow in a 90° degrees channel, a flow in a disc with a stirrer and a flow in an impinging jet. Only stationary, laminar flow with constant density is considered for these test cases.

6.1 Test case 1: Pipe flow in a unit cube

In this test case (X-stream reference XTC-35) the stationary incompressible Navier-Stokes equations are solved for a unit cube. We assume the flow has a constant viscosity.
The geometry is 3D and $x \times y \times z = 1$ m $\times$ 1 m $\times$ 1 m. The inlet is located at $x = 0$ m, the outlet at $x = 1$ m. The four remaining boundaries are all solid walls. The number of subdomains (blocks) can be chosen: 1, 2, 8 or 64.
For the velocity components uniform Dirichlet BCs are taken at the inlet ($u = 1\ ms^{-1}$, $v = w = 0\ ms^{-1}$) and homogeneous Neumann BCs at the outlet. At the walls no-slip conditions are taken.
Figure 6.1 shows the geometry, as well as the resulting velocity field for the case with 64 blocks.

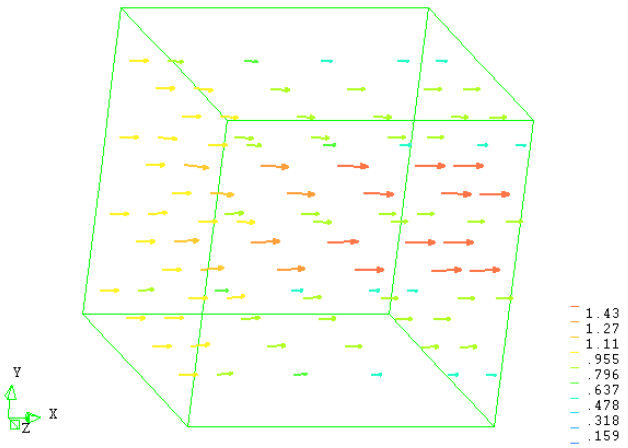

Fig. 6.1: Geometry and velocity field for a flow in a unit cube

6.2 Test case 2: Flow in a 90° degrees channel

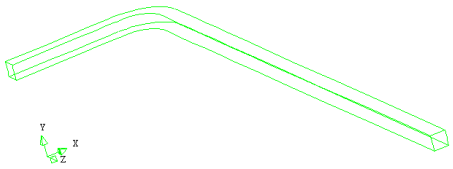

Fig. 6.2: Geometry of a 90° degrees channel

In this test case the stationary incompressible Navier-Stokes equations are solved for a 90° degrees channel.
The geometry is 3D and the channel has length $z = 1.6$ m, width $x = 0.4$ m and height $y = 0.04$ m (see Figure 6.2). The inlet is located at $x = 0$ m, the outlet at $z = 1.6$ m. The four remaining boundaries are all solid walls. The number of blocks is six.
For the velocity components uniform Dirichlet BCs are taken at the inlet and homogeneous Neumann BCs at the outlet. At the walls no-slip conditions are taken.
Figure 6.3 shows the resulting velocity and pressure field in a cross-section of the channel.

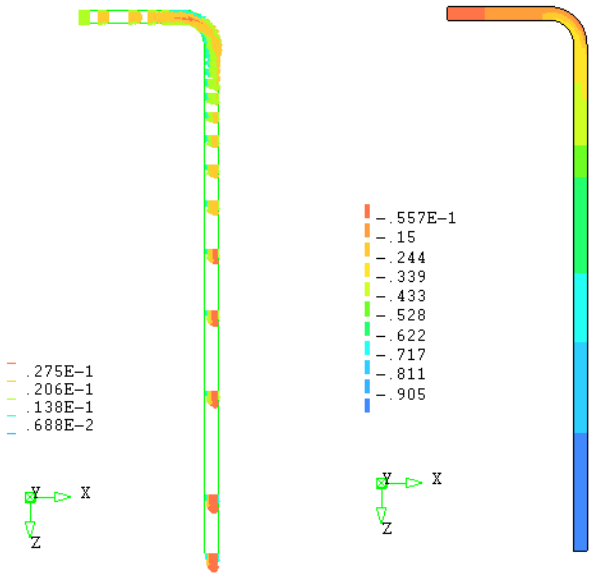

Fig. 6.3: The computed velocity field (left) and pressure field (right) in a cross-section of a 90° degrees channel

6.3 Test case 3: Flow in a disc with a stirrer

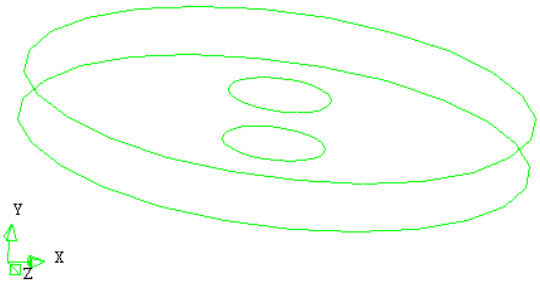

Fig. 6.4: Geometry of a disc

In this test case the stationary incompressible Navier-Stokes equations are solved for a disc with a so-called horizontal stirrer. Stirrers are often applied in the glass industry to enhance the mixing of the fluid.
The geometry is 3D and the disc has inner radius $R_{\text{in}} = 0.1$ m, outer radius $R_{\text{out}} = 0.5$ m and height $y = 0.1$ m, see Figure 6.4. There are two solid walls (at $r = 0.1$ m and at $r = 0.5$ m), the rest of the disc is 'open'. Four blocks are taken.
Note that the disc is *rotation-symmetrical* and that each horizontal cross-section (h constant) has the same velocity and temperature field.
At the walls uniform Dirichlet BCs are taken for the temperature. At the top ($y = 0.1$ m) and bottom ($y = 0$ m) of the disc uniform Neumann BCs are taken.
To simulate the effect of the stirrer, a body force is applied to the flow.
Figure 6.5 shows the velocity field and the temperature in a cross-section of the disc.

6.4 Test case 4: Flow in an impinging jet

In this test case the stationary incompressible Navier-Stokes equations are solved for an impinging jet.
The geometry is 3D, a cube with $x \times y \times z = 0.1$ m $\times$ 0.1 m $\times$ 0.1 m. The inlet is a 0.02 m $\times$ 0.02 m square located in the middle of the top of the cube and the four sides of the cube are the outlets (so at $x = 0$ m, $x = 0.1$ m, $y = 0$ m and $y = 0.1$ m). The bottom of the cube ($z = 0$ m) and the top of the cube ($z = 0.1$ m) minus the inlet, are solid walls. Three blocks are taken.
For the velocity components homogeneous Dirichlet BCs are taken at the walls and uniform Dirichlet BCs at the inlet. For the pressure homogeneous Dirichlet BCs are taken at the outlets.
Figure 6.6 shows the velocity field and the temperature in a cross-section of the cube.

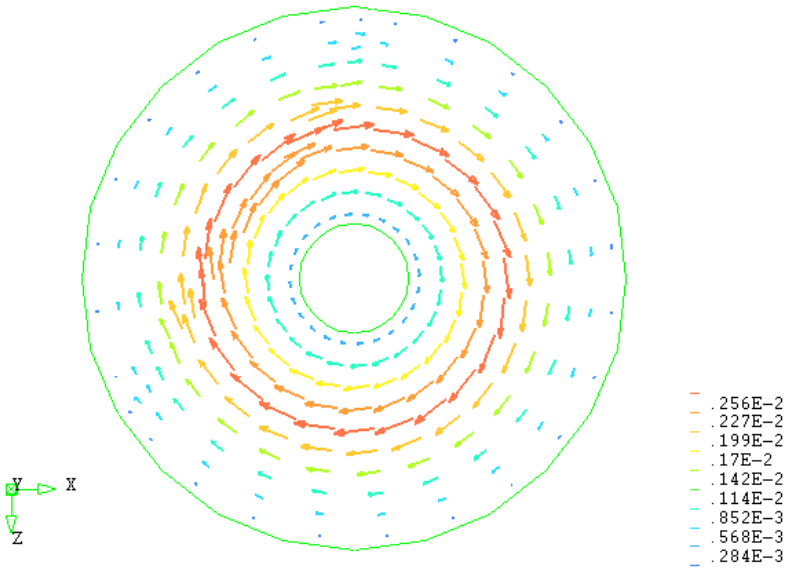

Fig. 6.5: The resulting velocities (arrows) and temperatures (colors) in a cross-section of a disc

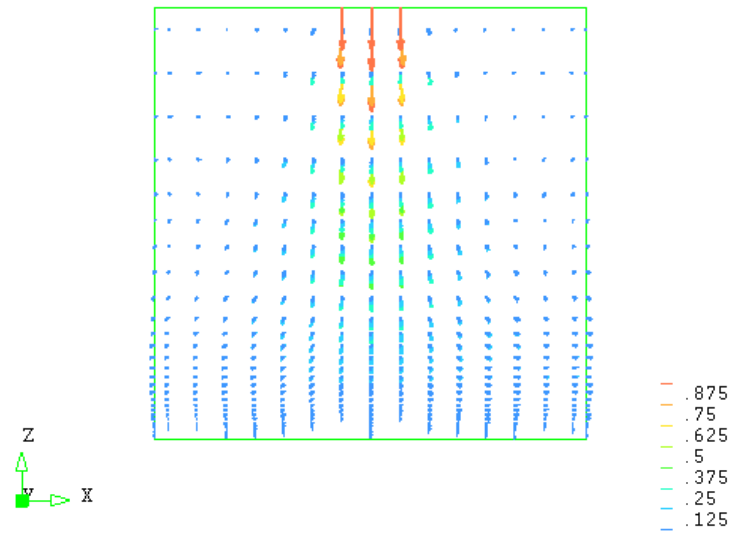

Fig. 6.6: The resulting velocities (arrows) and temperatures (colors) in a cross-section of an impinging jet

7. NUMERICAL EXPERIMENTS WITH DEFLATION

In this chapter we want to gain insight into the DD methods, as described earlier. Especially the choice of the deflation vectors (the columns of Z, see Section 3.3) will be investigated. Therefore tests in MATLAB and X-stream have been done. MATLAB is used since it has a huge number of build-in routines for matrix analysis, and programming in MATLAB is rather straightforward.

The structure of this chapter is as follows. First the choice and description of deflation vectors are discussed in Section 7.1. Then the tests and their results are described in Sections 7.2 (MATLAB) and 7.3 (X-stream). Partitioning of the subdomains is treated in Section 7.4. When using deflation, an LU decomposition of the matrix $Z^T AZ$ has to be made. The accuracy of this LU decomposition is important, this is treated in Section 7.5.

7.1 Deflation vectors

This section begins with the choice of deflation vectors: the requirements for these vectors are given in Subsection 7.1.1. After this, three different kinds of deflation vectors are described in Subsection 7.1.2.

7.1.1 Requirements for deflation vectors

Choosing deflation vectors for obtaining an efficient solution method is not a trivial task. Several conditions have to be taken into account, some of which are already mentioned in Section 3.3. Consider Y chosen equal to Z. We want the matrix Z to satisfy the following properties (see Vuik *et al.* [30]):

1. The matrix Z should be problem independent and inexpensive to construct.
2. The span of the columns of Z should be close to the eigenspace corresponding with the smallest eigenvalues of PA.
3. Generalization to 3D should be possible.
4. The matrix Z should be chosen such that $E = Z^T AZ$ is non-singular, because systems involving E are computed by a direct method.
5. The gain in iterations for this choice of Z should result in lower wall-clock times.

Property 2 is difficult to obtain, because little theory is developed on how the choice of Z is related to the spectrum of PA. However, for the case that A is SPD, a few bounds for the eigenvalues have been proved, see Nicolaides [11].

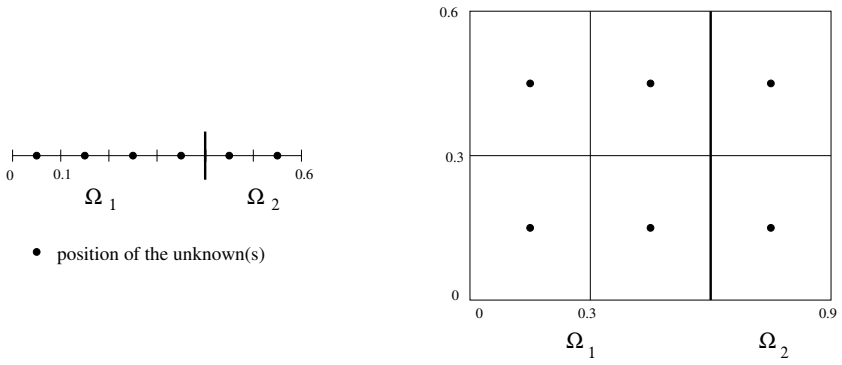

Fig. 7.1: 1D and 2D grid, both with two subdomains

Considering property 5 the following can be stated. The wall-clock time depends on more factors than the choice of Z: the convergence rate of the iterative solution method, the number of deflation vectors, the number of grid cells per subdomain and the solution method used to solve the system involving E also play a role. However, it is still important to choose Z as good as possible.

7.1.2 Description of deflation vectors

In this subsection, we will consider three different kinds of deflation vectors: constant deflation vectors and linear deflation vectors, based on grid numbering and based on grid coordinates. This results in three possibilities for the matrix Z used in deflation methods: Z_{CD} for constant deflation, $Z_{CLD-ijk}$ for constant linear deflation based on grid numbering and $Z_{CLD-cartesian}$ for constant linear deflation based on grid coordinates. These three deflation matrices will be described with help of two examples, a 1D grid with two subdomains and a 2D grid with two subdomains, see Figure 7.1. After the description of the deflation vectors and matrices, CLD-ijk and CLD-cartesian are compared.

Constant deflation vectors

For each subdomain, exactly one deflation vector is defined having constant elements in the grid points on the corresponding subdomain, and zero elements in the grid points on the other subdomain(s). So constant deflation vectors $\mathbf{z}_m$ are chosen in the following way:

$$\begin{aligned} \mathbf{z}_m(i) &= 1,\ \mathbf{x}_i \in \Omega_m, \\ \mathbf{z}_m(i) &= 0,\ \mathbf{x}_i \notin \Omega_m. \end{aligned}$$

For both the examples (1D and 2D) in Figure 7.1 this gives the following defla-

tion matrix:

$$Z_{CD} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}. \tag{7.1}$$

The value 1 in Z_{CD} can be replaced by a different non-zero value, since the matrix PA only depends on span(Z). Generalization to the 3D case is straightforward.

Linear deflation vectors based on grid numbering

A linear deflation vector based on grid numbering is defined on each subdomain by increasing each element linearly for increasing grid points. For a 1D case the vectors $\mathbf{z}_m$ are chosen in the following way:

$$\begin{aligned} \mathbf{z}_m(i) &= 0,\ \mathbf{x}_i \notin \Omega_m, \\ \mathbf{z}_1(i) &= i,\ \mathbf{x}_i \in \Omega_1, \\ \mathbf{z}_m(i) &= i - \sum_{j=1}^{m-1} |\Omega_j|,\ \mathbf{x}_i \in \Omega_m,\ \mathrm{m} > 1, \end{aligned}$$

with $|\Omega_j|$ the number of cells in subdomain j. Here we assume the grid is structured and has a logical numbering.

The columns of the deflation matrix span the space consisting of both the constant and the linear vector. So it combines the constant deflation vectors and the linear deflation vectors based on grid numbering.
For the 1D example (see Figure 7.1) this gives the following deflation matrix:

$$Z_{CLD-ijk} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ 1 & 4 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}. \tag{7.2}$$

Generalization to the 2D and 3D case is not so straightforward as with constant deflation. In two dimensions Z consists of three vectors per subdomain: one constant vector and one linear vector for each of the two dimensions. In three dimensions Z consists of four vectors per subdomain: one constant vector and three linear vectors.

Consider the following 2D example, given in Figure 7.2: one domain with four cells, the cells numbered as shown in the figure. The constant deflation vector is a vector with four ones. For making the linear deflation vectors both numberings of the cells are used: the 'single' numbering for the sequence of the numbers in the vector, the (i,j)-numbering for the values of the numbers in the vector. The linear deflation vector in horizontal direction is given by $(1\ 2\ 1\ 2)^T$ and the

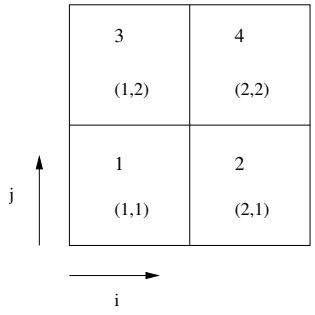

Fig. 7.2: 2D grid with one domain and cells numbered in two ways

linear deflation vector in vertical direction is given by $(1\ 1\ 2\ 2)^T$. The matrix $Z_{CLD-ijk}$ is now given by:

$$Z_{CLD-ijk} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 2 \end{pmatrix}. \tag{7.3}$$

Linear deflation vectors based on grid coordinates

A linear deflation vector based on grid coordinates is defined on each subdomain by using the positions of the cell centers. For a 1D case the vectors $\mathbf{z}_m$ are chosen in the following way:

$$\begin{aligned} \mathbf{z}_m(i) &= \mathbf{x}_i,\ \mathbf{x}_i \in \Omega_m, \\ \mathbf{z}_m(i) &= 0,\ \mathbf{x}_i \notin \Omega_m. \end{aligned}$$

Just as with linear deflation based on the grid, for the deflation matrix the constant deflation vectors and the linear deflation vectors based on grid coordinates are combined.
For the 1D example (see Figure 7.1) this gives the following deflation matrix (the cells have length 0.1):

$$Z_{CLD-cartesian} = \begin{pmatrix} 1 & 0.05 & 0 & 0 \\ 1 & 0.15 & 0 & 0 \\ 1 & 0.25 & 0 & 0 \\ 1 & 0.35 & 0 & 0 \\ 0 & 0 & 1 & 0.45 \\ 0 & 0 & 1 & 0.55 \end{pmatrix}. \tag{7.4}$$

For the 2D case Z has three vectors per subdomain: one constant vector and two linear vectors: one for the x-coordinates and one for the y-coordinates of the cell centers. For the 3D case Z has four vectors per subdomain.

Non-uniform grids

In the given examples uniform grids are used. Generalization to non-uniform grids is straightforward. For constant deflation vectors and linear deflation

Fig. 7.3: 1D grid with two subdomains and two cell sizes

vectors based on grid numbering nothing changes. Only linear deflation vectors based on grid coordinates are different. Consider a 1D example, with grid size 0.1 in the first subdomain and grid size 0.2 in the second subdomain, see Figure 7.3.
Now the deflation matrix is:

$$Z_{CLD-cartesian} = \begin{pmatrix} 1 & 0.05 & 0 & 0 \\ 1 & 0.15 & 0 & 0 \\ 1 & 0.25 & 0 & 0 \\ 1 & 0.35 & 0 & 0 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0 & 1 & 0.7 \end{pmatrix}. \tag{7.5}$$

Comparison between CLD-ijk and CLD-cartesian

We have now constructed deflation matrices Z. Note that the matrix $Z_{CLD-cartesian}$ in (7.4) looks like $Z_{CLD-ijk}$ in (7.2). It is useful to investigate whether the results of CLD-ijk and CLD-cartesian are the same under certain conditions.

In algorithms that use deflation, $P = I - AZ(Y^TAZ)^{-1}Y^T$ and $Q = I - Z(Y^TAZ)^{-1}Y^TA$ are constructed and used. We choose Y equal to Z, as stated earlier. The following Lemma can be applied:

Lemma 7.1.
Let $A \in \mathbb{R}^{n\times n}$ be non-singular. Let $Z_1 \in \mathbb{R}^{n\times r}$ and $Z_2 \in \mathbb{R}^{n\times r}$ with rank(Z_1) =rank(Z_2) = r. Define $E_1 = Z_1^TAZ_1$ and $E_2 = Z_2^TAZ_2$. If Im(Z_1)=Im(Z_2), then

$$(I - AZ_1E_1^{-1}Z_1^T)A = (I - AZ_2E_2^{-1}Z_2^T)A.$$

Proof.
Since Im(Z_1)=Im(Z_2), there exists a non-singular matrix $M \in \mathbb{R}^{r\times r}$ such that $Z_1 = Z_2M$. Hence

$$\begin{aligned} Z_2{E_2}^{-1}{Z_2}^T - Z_1{E_1}^{-1}{Z_1}^T &= Z_2({E_2}^{-1} - M{E_1}^{-1}M^T){Z_2}^T \\ &= Z_2{E_2}^{-1}(I - E_2M{E_1}^{-1}M^T){Z_2}^T \\ &= Z_2{E_2}^{-1}(I - E_2M(M^TE_2M)^{-1}M^T){Z_2}^T \\ &= Z_2{E_2}^{-1}(I - E_2MM^{-1}{E_2}^{-1}M^{-T}M^T){Z_2}^T \\ &= 0. \end{aligned}$$

A consequence of this Lemma is that, if A is SPD and **if each subdomain has uniform cells**, then CLD-ijk and CLD-cartesian produce the same result,

because the two methods have the same matrix P and Q. Take for example the matrices (7.2) and (7.5) (these matrices belong to a grid with two subdomains, each subdomain has its own cell size, see Figure 7.3). It is easy to show that the span of $Z_{CLD-cartesian}$ is equal to the span of $Z_{CLD-ijk}$. The first and third column of the matrices are equal. Furthermore

$$\begin{pmatrix} 0.05 \\ 0.15 \\ 0.25 \\ 0.35 \\ 0 \\ 0 \end{pmatrix} = 0.1 \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 0 \\ 0 \end{pmatrix} - 0.05 \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} \tag{7.6}$$

and

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0.5 \\ 0.7 \end{pmatrix} = 0.3 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} + 0.2 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 2 \end{pmatrix}. \tag{7.7}$$

In the next section one test is done for subdomains with uniform cells, as an example of Lemma 7.1. The other tests are done for cases with different cell sizes within a subdomain.

7.2 Tests in MATLAB

In MATLAB two 1D testproblems have been used:

- **Testproblem 1** (Poisson problem, Dirichlet BCs):

  $$\frac{d^2\varphi}{dx^2} = x \sin x, \tag{7.8}$$

  with boundary conditions $\varphi(0) = \varphi(\pi) = 0$.

- **Testproblem 2** (convection-diffusion problem, Dirichlet BCs):

  $$\frac{d\varphi}{dx} - 0.001\frac{d^2\varphi}{dx^2} = 0, \tag{7.9}$$

  with boundary conditions $\varphi(0) = 0$ and $\varphi(1) = 1$.

These problems have been tested on two grids:

- **Grid 1** (see Figure 7.4):
  A 1D domain with n cells: in the first half of the domain the cells have length h_1, in the second half of the domain the cells have length h_2.

- **Grid 2** (see Figure 7.4):
  A 1D domain with n cells, cell i has length $h(i) = \alpha * (k)^i$. α depends on the length of the domain and it is convenient to choose k approximately 0.9.

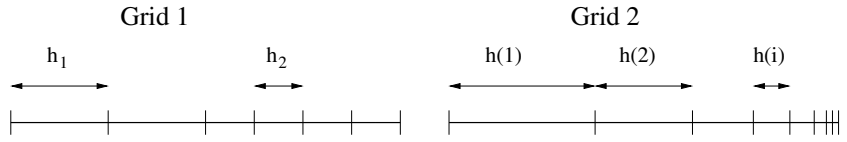

Fig. 7.4: Grids for the tests in MATLAB

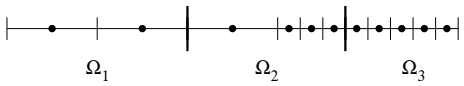

Fig. 7.5: Grid 1 with three subdomains

Both testproblems are discretized with central differences and solved with PGCR and DPGCR. PCG and DPCG are not used because the second testproblem is non-symmetric. The diagonal matrix is used as a preconditioner. The startvector is the zero-vector and the stopcriterion is:

$$\frac{\|b - Ax\|}{\|b\|} < 0.0001 \tag{7.10}$$

The tests consist of trying PGCR and DPGCR with three deflation matrices (CD, CLD-ijk and CLD-cartesian) with various values for n and a various number of subdomains.

In Lemma 7.1., we have proved that if each subdomain has uniform cells, then CLD-ijk and CLD-cartesian produce the same result. Therefore, we will perform one test in MATLAB where this is the case: testproblem 1 with grid 1, where grid 1 has two subdomains that each cover exactly half of the domain (as in Figure 7.3). Furthermore, $h_1 = 3h_2$ is chosen. The number of iterations for CLD-ijk and CLD-cartesian for a different number of cells n is:

n	CLD-ijk	CLD-cartesian
100	72	72
200	145	145
300	218	218

This shows what we expected considering Lemma 7.1. Because the cases with uniform cells in each subdomain are not so interesting, in all other tests we will consider domains which have at least one subdomain with non-uniform cell-sizes. See for example grid 1 with three subdomains in Figure 7.5. One can see that the first and third subdomain have uniform cell sizes, while the second subdomain has cells with two different sizes.

7.2.1 Results

In this subsection the results of the tests in MATLAB are given. This means that the number of iterations is given for every test. First, several tables with test results are given, after that some conclusions are drawn. Only in the first

table results for PGCR are given. More tests for PGCR have been performed, but they show nothing new.

Results for testproblem 1, grid 1 with $h_1 = 3h_2$:

(D)PGCR, one subdomain:

n	PGCR	CD	CLD-ijk	CLD-cartesian
100	100	92	97	97
200	200	181	195	194
300	300	264	292	292

DPGCR, three subdomains:

n	CD	CLD-ijk	CLD-cartesian
108	68	56	43
204	128	107	81
300	188	157	120

DPGCR, five subdomains:

n	CD	CLD-ijk	CLD-cartesian
100	39	30	30
200	76	60	61
300	113	91	91

Results for testproblem 2, grid 1 with $h_1 = 3h_2$:

DPGCR, one subdomain:

n	CD	CLD-ijk	CLD-cartesian
100	99	98	98
200	159	158	158
300	219	218	218

DPGCR, three subdomains:

n	CD	CLD-ijk	CLD-cartesian
108	82	68	68
204	118	109	109
300	170	152	153

DPGCR, five subdomains:

n	CD	CLD-ijk	CLD-cartesian
100	66	58	57
200	88	82	80
300	124	111	112

Instead of taking $h_1 = 3h_2$, another ratio between h_1 and h_2 can be chosen. This has also been tested, but shows nothing new. The results of testproblem 2 are very similar to the results of testproblem 1 (except for the grid with three subdomains), so for grid 2 only the results for testproblem 1 are given.

For testproblem 1, grid 1 and three subdomains CLD-cartesian shows a much better performance than CLD-ijk, whereas for an other number of subdomains the results are similar. This could have to do with the number of subdomains with different cell sizes. In case of three subdomains, the first and third subdomain have equal cells within the subdomain. The second subdomain contains two different cell sizes. In case of five subdomains, only one out of the five subdomains has different cell sizes, which is less than one out of three subdomains. This could possibly increase the difference between CLD-ijk and CLD-cartesian.
However, this is not a satisfactory explanation. One outcome that can not be explained by this theory is the case with one domain: on basis of the theory above one would expect a large difference between the performances of CLD-ijk and CLD-cartesian. Another thing that can not be explained are the results of testproblem 2: the case with three subdomains does not show much difference between CLD-ijk and CLD-cartesian there.

Results for testproblem 1, grid 2 with $\alpha = 0.33$ and $k = 0.9$:

DPGCR, one subdomain:

n	CD	CLD-ijk	CLD-cartesian
50	48	48	47
65	64	62	63

DPGCR, three subdomains:

n	CD	CLD-ijk	CLD-cartesian
51	46	32	13
72	63	50	20

DPGCR, five subdomains:

n	CD	CLD-ijk	CLD-cartesian
50	36	18	7
70	55	30	11

α, k and n have to be chosen carefully. If α is too large, n has to be small, otherwise negative cell lengths occur. If k is small and n is large, the cell sizes get very small and MATLAB has trouble calculating them.

7.2.2 Conclusions

From the tests performed in MATLAB, some conclusions can be drawn:

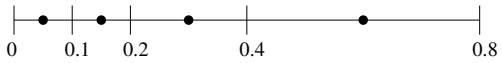

Fig. 7.6: 1D grid with non-uniform cell-sizes

- Solving a problem with PGCR always means more iterations than solving the problem with DPGCR.
- CLD almost always needs less iterations than CD. Only for testproblem 1 with one subdomain CD is slightly better.
- More subdomains means less iterations. This can be explained as follows: more subdomains means less grid points (unknowns) per subdomain. In general, less unknowns means less iterations. When you keep increasing the number of subdomains, in the end each subdomain will have **one** unknown. It is obvious that in this case the problem is solved directly. Drawback of increasing the subdomains is that each iteration becomes more expensive. For every problem, there has to be an optimal number of subdomains such that the total wall-clock time is as low as possible.
- For grid 1, there is very little difference between CLD-ijk and CLD-cartesian. Only testproblem 1 with three subdomains shows a much better performance of CLD-cartesian.
- For grid 2, CLD-cartesian performs better than CLD-ijk.

We are especially interested in the difference in performance between CLD-ijk and CLD-cartesian. The last two conclusions say something about these two methods. The difference between CLD-ijk, CD and iterative methods without deflation is already treated in Verkaik [19].
Using common sense, one would expect that CLD-cartesian performs better than CLD-ijk, because for the vectors of CLD-ijk the numbering of the cells is important, while for the vectors of CLD-cartesian the positions of the unknowns are important.

Consider Figure 7.6. The CLD-ijk vector for this grid is $(1\ 2\ 3\ 4)^T$ and the CLD-cartesian vector is $(0.05\ 0.15\ 0.3\ 0.6)^T$. The CLD-cartesian vector better represents the position of the unknowns and the sizes of the cells than the CLD-ijk vector. An explanation for the fact that grid 1 does not show much difference can be that there are only two different cell-sizes, and h_1 is 'only' three times as large as h_2. In grid 2 all the cells have different sizes and the difference between the smallest and largest cell size is very big.

So much for the common sense. A more mathematical explanation can be given considering the effective condition numbers of the matrices $P_{CLD-ijk}A$ and $P_{CLD-cartesian}A$: these are the matrices used in the deflation algorithms. We look at the effective condition number because the matrix PA is singular (the effective condition number is the maximum eigenvalue divided by the minimum non-zero eigenvalue, given the problem is symmetric).
For example, in the table given below some effective condition numbers are

given together with the number of iterations for testproblem 2 and grid 1:

Testproblem 2, grid 1 with $h_1 = 3h_2$:

DPGCR, five subdomains:

n	#iter CLD-ijk	#iter CLD-cart	eff cond no CLD-ijk	eff cond no CLD-cart
100	58	57	17.5	17.4
200	82	80	20.4	19.0
300	111	112	36.3	36.3

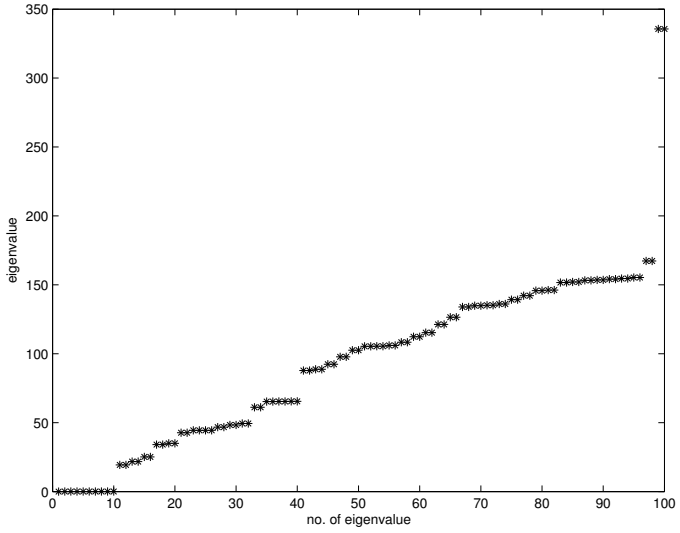

Fig. 7.7: The eigenvalues of CLDGCR-cartesian for $n = 100$, testproblem 2, grid 1

The eigenvalues of PA for $n = 100$ with CLD-cartesian are given in Figure 7.7. For other values of n, other numbers of domains and CLD-ijk instead of CLD-cartesian the eigenvalues are distributed in the same way. You can see that the value of the effective condition number corresponds with the number of iterations required for convergence. Because five subdomains are considered, there are ten deflation vectors, this explains the ten eigenvalues with value zero you see in Figure 7.7.

It is interesting to consider a case where the number of iterations of CLD-ijk and CLD-cartesian differ a lot. This is the case for testproblem 1 in combination with grid 2. Unfortunately, it is difficult to calculate and compare the effective condition numbers, because there are a lot of (very) small eigenvalues and some very large ones. Therefore the eigenvalues for testproblem 1, grid 2 with $\alpha = 0.33$ and $k = 0.9$, $n = 50$ and five subdomains are plotted on log scale in Figure 7.8. The ten eigenvalues with value zero are left out of the figure. The eigenvalues for other numbers of domains and other values of n are distributed in the same way. Now the effective condition numbers can be calculated. It

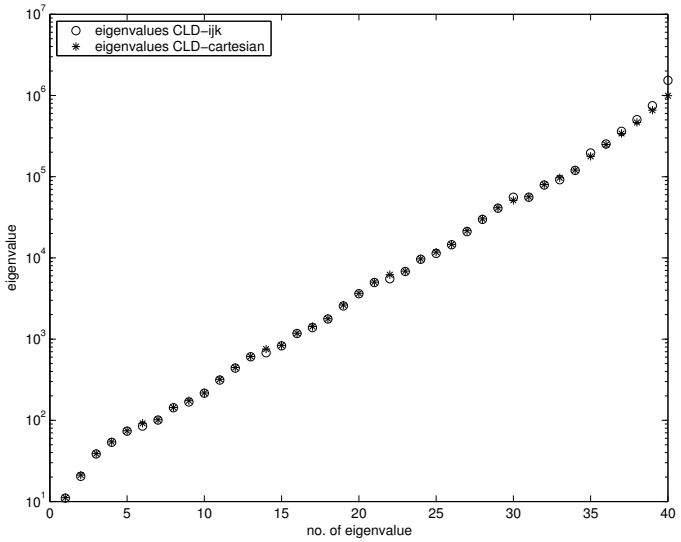

Fig. 7.8: The eigenvalues of CLDGCR for $n = 50$, testproblem 1, grid 2

turns out that CLD-ijk has an effective condition number of $1.4 \cdot 10^5$ and CLD-cartesian has an effective condition number of $8.9 \cdot 10^4$. This partly explains the difference in number of iterations needed for convergence (CLD-ijk needs 18 iterations, CLD-cartesian needs 7 iterations).

7.3 Tests in X-stream

In X-stream three test cases have been used. The stationary incompressible Navier-Stokes equations are solved for all three cases. The three test cases are described in Sections 6.2, 6.3 and 6.4. In Subsection 7.3.1 it is made clear what we know already and what needs to be tested. The results of the test cases can be found in Subsection 7.3.2 and the conclusions in Subsection 7.3.3.

7.3.1 Tests

In X-stream, the test cases described in Chapter 6 can be solved with different solvers. Important output of the test cases are the number of outer iterations, the residuals and the wall-clock times. Here is a survey of the things that have been varied for the test cases. The default values are pointed out and remarks are made about the usefulness of varying the parameters considering the output of the test cases.

- *Solver*: the solvers that can be chosen are SIP, SPTDMA, CG and GCR. The last two can be used with deflation: CD, CLD-ijk and CLD-cartesian. Deflation can be used for pressure or energy, for example. We only consider SIP and GCR (without and with deflation for the pressure), with emphasis on (D)GCR. More on the performance of SPTDMA can be found in Verkaik [19]. The SIP method is most commonly used in X-stream.

- *Underrelaxation*: the underrelaxation parameters of the velocities and the pressure can be varied. Verkaik [19] has done research on choosing the underrelaxation parameters. We will use the default values in X-stream.
- *Block level*: the block level (the refinement level of the blocks) has default value 1. The block level can be set to 2 but this shows nothing new concerning the differences between the solvers. Also, the program needs much more time to reach convergence. Therefore we will only work with block level 1.
- *SIMPLE stabilizer iteration*: Convergence results on varying SSI can be found in Verkaik [19]. For each test case, we use the default value X-stream has set.
- *Inner iterations*: for most problems, the default values for the number of inner iterations are 5 for the velocities and 15 for the pressure and energy (if the energy equation is solved). We will vary the number of inner iterations for the pressure.

To sum up the above: we vary the number of inner iterations of the pressure to evaluate the SIP solver and (D)GCR. Important forms of output are the number of outer iterations, wall-clock time and residuals. Because Verkaik [19] has already done a lot of research on the different solvers, except for DGCR with CLD-cartesian, emphasis will lay on the difference in performance between DGCR with CLD-ijk and DGCR with CLD-cartesian. (D)CG is not considered because (D)GCR is more general, applicable also to non-symmetric problems.

For solving all variables, the additive Schwarz method is used with inaccurate subdomain solution obtained by SIP steps. We restrict ourselves to solving the subdomain problems using one SIP iteration and refer to this method as SIP(1).

7.3.2 Results

Test case 2 (Flow in a 90° degrees channel)

Figure 7.9 shows the residuals for the methods SIP(1) and (D)GCR for 30 inner iterations. One outer iteration is done. The residual is given by the following: for each cell the mass that enters the cell and the mass that leaves the cell are calculated. For each cell the difference between these two numbers is calculated. The sum over all the cells in the grid of these absolute differences is the residual.

This figure shows a very good performance of CLDGCR-cartesian. After about 12 inner iterations CLDGCR-ijk has the second best residual. After more than 30 iterations the residuals remain approximately constant.

Figure 7.10 shows the number of outer iterations that is needed for convergence for a certain number of inner iterations. Figure 7.11 shows the time needed for convergence for a certain number of inner iterations.

For three or more inner iterations, CLDGCR-ijk and CLDGCR-cartesian have the same number of outer iterations. For six or more inner iterations, CDGCR has the same number of outer iterations as well.

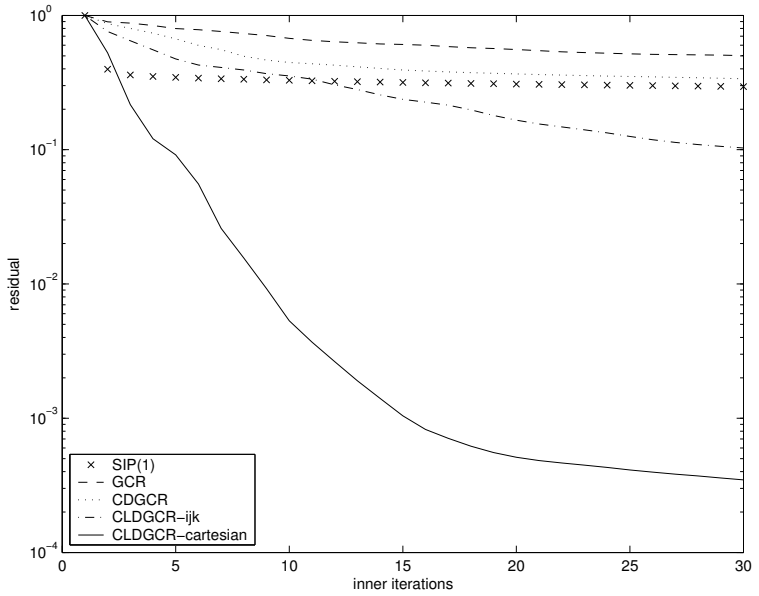

Fig. 7.9: The residuals after one outer iteration for a certain number of inner iterations for test case 2

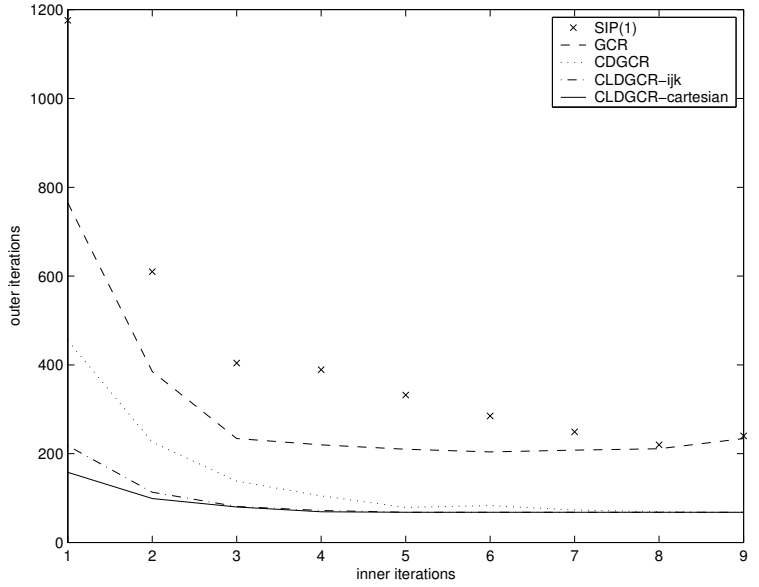

Fig. 7.10: The number of outer iterations needed for convergence for test case 2

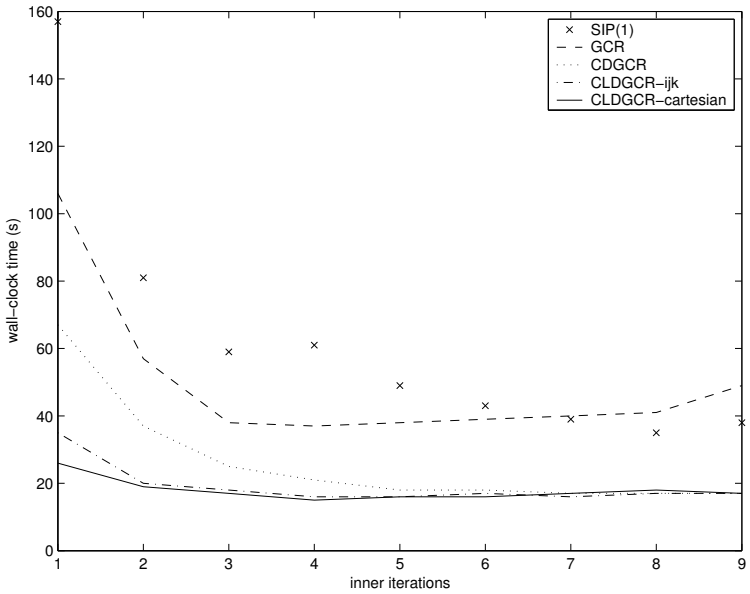

Fig. 7.11: Time (in seconds) needed for convergence for test case 2

SIP(1) always needs more iterations. Especially for a very low number of inner iterations, the gain by using CLDGCR-cartesian instead of SIP(1) is very large. The table given below illustrates this. The number of outer iterations SIP(1) needs is divided by the number of outer iterations an other method needs.

# inner it.	$\frac{\text{SIP(1)}}{\text{GCR}}$	$\frac{\text{SIP(1)}}{\text{CDGCR}}$	$\frac{\text{SIP(1)}}{\text{CLDGCR-ijk}}$	$\frac{\text{SIP(1)}}{\text{CLDGCR-cart}}$
1	1.5	2.6	5.4	7.4
2	1.6	2.7	5.4	6.2
6	1.4	3.4	4.2	4.2
9	1.0	3.6	3.6	3.6

You can see that for one inner iteration, CLDGCR-cartesian needs about seven times less outer iterations, while for nine inner iterations, CLDGCR-cartesian needs 'only' 3.6 times less outer iterations.
Figure 7.11 shows almost the same picture as Figure 7.10. Only after about six inner iterations, the time to convergence increases slightly for (D)GCR. A logical explanation for this is the increasing number of inner iterations. (when the outer iterations are constant and the inner iterations increase, the total number of iterations also increases). This pleads for choosing the number of inner iterations not too large. To give an idea of the time you gain using (D)GCR instead of SIP(1), the table below indicates the time (D)GCR needs compared to the time SIP(1) needs to converge.

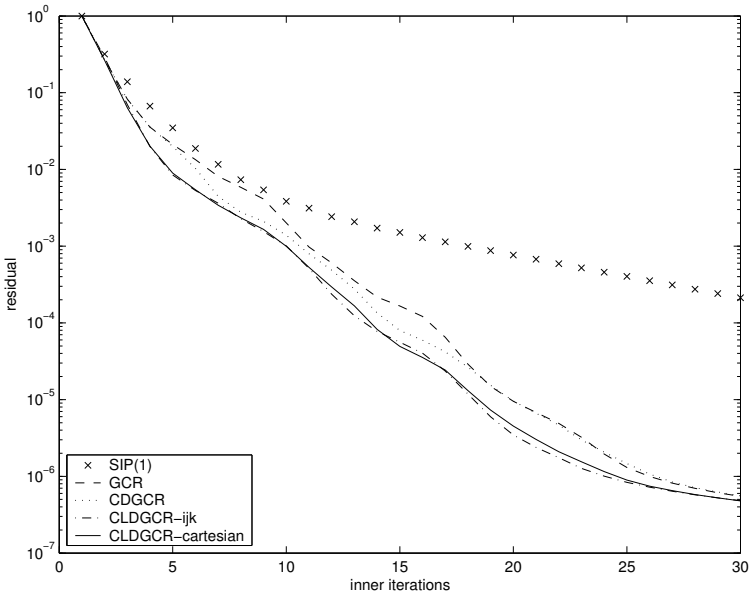

Fig. 7.12: The residuals after one outer iteration for a certain number of inner iterations for test case 3

# inner it.	$\frac{\text{SIP(1)}}{\text{GCR}}$	$\frac{\text{SIP(1)}}{\text{CDGCR}}$	$\frac{\text{SIP(1)}}{\text{CLDGCR-ijk}}$	$\frac{\text{SIP(1)}}{\text{CLDGCR-cartesian}}$
1	1.5	2.3	4.5	6.0
2	1.4	2.2	4.1	4.3
6	1.1	2.4	2.5	2.7
9	0.8	2.2	2.2	2.2

The table above is correct, but not completely fair. Normally, you would not use the same number of inner iterations for the different methods. Therefore we compare the times to convergence of the different methods, given that the number of inner iterations is optimal for each method. For SIP(1) we use 12 inner iterations, for GCR 4, for CDGCR 7, for CLDGCR-ijk 4 and for CLDGCR-cartesian 4. Now the time you can gain by using (D)GCR instead of SIP(1) is given in the table below.

$\frac{\text{SIP(1)}}{\text{GCR}}$	$\frac{\text{SIP(1)}}{\text{CDGCR}}$	$\frac{\text{SIP(1)}}{\text{CLDGCR-ijk}}$	$\frac{\text{SIP(1)}}{\text{CLDGCR-cartesian}}$
0.9	2	2.1	2.3

This table shows that applying a deflation method to GCR can make the time to convergence twice as fast.

Test case 3 (Flow in a disc with a stirrer)

Figure 7.12 shows the residuals for the solvers SIP(1) and (D)GCR for 30 inner iterations. One outer iteration is done.
The figure shows not much difference between GCR and DGCR. Clearly, SIP(1)

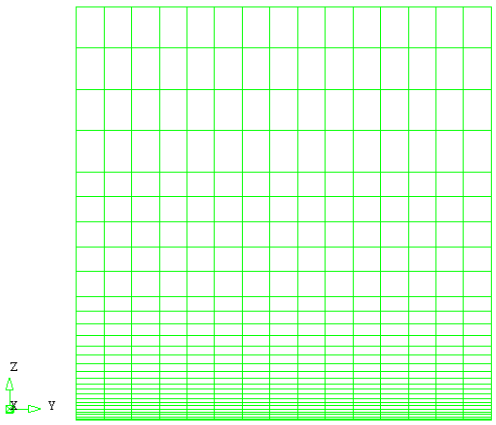

Fig. 7.13: Cells in a cross-section of the cube of test case 4

has larger residuals than the other solvers. The little difference between GCR and DGCR can be explained by (as already stated at the description of the test case) the rotation-symmetry of the disc and the fact that every horizontal cross-section of the disc has the same velocity and temperature field. What you actually do is solving a problem along a line of 0.4 m and then copy this to the rest of the disc. This 'line' has uniform cell size, so this explains the very similar performances of GCR, CDGCR, CLDGCR-ijk and CLDGCR-cartesian. Because this test case is clearly not interesting if you want to see the difference between CLDGCR-ijk and CLDGCR-cartesian we do not proceed with this case.

Test case 4 (Flow in an impinging jet)

Tests in MATLAB showed a very good performance of CLD-cartesian compared to the other methods, for a grid with much difference in the cell sizes. Therefore, test case 4 is tested with the grid as in Figure 7.13: the cells become smaller toward the bottom of the cube. We expect to see differences in performance of the different solvers for this test case.

Figure 7.14 shows the residuals for the solvers SIP(1) and (D)GCR for 30 inner iterations. DGCR needs less than 30 iterations as you can see in the figure. Clearly, SIP(1) has larger residuals than the other solvers. CLDGCR-ijk and CLDGCR-cartesian perform equally well.

Varying the number of inner iterations has (almost) no influence on the number of outer iterations. There is little difference between the methods and every method itself shows only small variation in the number of outer iterations. Unfortunately, this test case does not show the same as the testproblems in MATLAB did.

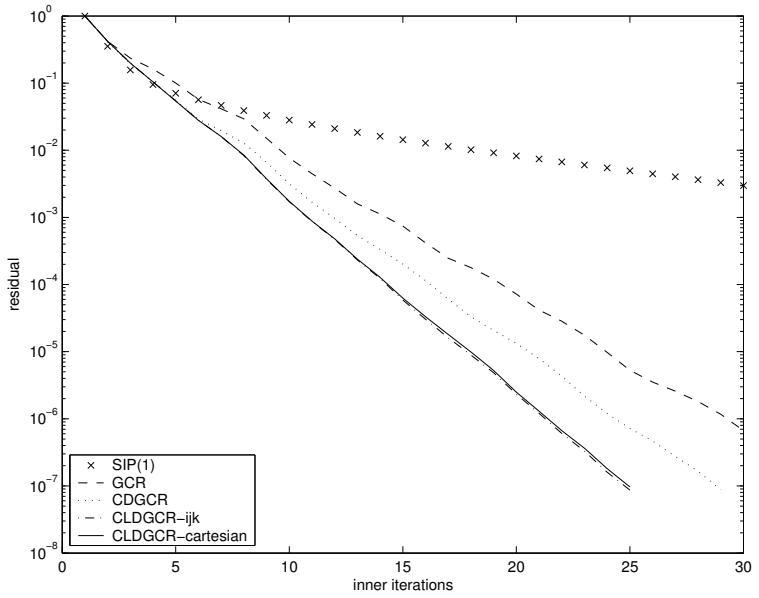

Fig. 7.14: The residuals after one outer iteration for a certain number of inner iterations for test case 4

7.3.3 Conclusions

From the above results some conclusions can be derived:

- CLDGCR-cartesian always needs less (or just as much) iterations than the other methods evaluated.
- CLDGCR-cartesian has always smaller (or just as large) residuals than the other methods evaluated.
- One should not choose the number of inner iterations too large. The advantage of choosing a large number of inner iterations, is that you are sure the number of outer iterations is as small as possible. The disadvantage is that a larger number of inner iterations costs more time than a smaller number of inner iterations, given the number of outer iterations stays (approximately) the same. Every problem has an optimum in balancing the number of outer and inner iterations.

7.4 Subdomain partitioning

Apart from the test cases described in Section 7.3, other cases have been tested with deflation in X-stream, small cases as well as larger ones. In general, the results of these cases resemble the results and conclusions given in Section 7.3. Only one new issue came up during the tests: the number of subdomains that is used. This will be explained below.

It is clear that when a certain 3D geometry is divided into n subdomains, the matrix Z_{CD} has n columns and Z_{CLD} has $4n$ columns. With m the number of cells, Z_{CD} is an $m \times n$ -matrix and Z_{CLD} is an $m \times 4n$ -matrix. In X-stream, the matrix $E = Z^T AZ$ is used. A is an $m \times m$ -matrix, so E_{CD} is an $n \times n$ -matrix and E_{CLD} a $4n \times 4n$ -matrix.
So a large number of subdomains causes a large matrix E, especially E_{CLD} is very large. For example, one of the test cases was a case with 600 subdomains, so then E_{CLD} is a 2400×2400 -matrix.
The problem with E being a large matrix are the calculations done with E. In X-stream, deflation is implemented in a way that is suitable for small matrices E: $E\mathbf{x} = \mathbf{y}$ is not solved efficiently for x, because no use of the sparseness of E is made. When E is small, the increase of the total computation time by this inefficiency is negligible. When E is large, the computation time can increase a lot.
This was shown by the test case with 600 subdomains. This case has run with GCR, CDGCR and CLDGCR-cartesian and with one, three and ten outer iterations. The number of inner iterations is set optimal for each solution method. The residuals of the velocities and pressure are roughly the same for the three methods (CLDGCR has somewhat smaller residuals, but it is not a large difference). The computation times differ a lot. For a small number of outer iterations, CDGCR needs about three times more CPU time than GCR, CLDGCR needs about three hundred times more CPU time than GCR.

As stated above, $E\mathbf{x} = \mathbf{y}$ is not solved efficiently. First an LU decomposition of E is made, then this LU decomposition is used for solving $E\mathbf{x} = \mathbf{y}$. Making the LU decomposition of E costs a lot of time: for this case, CDGCR needs ten to fifteen seconds to make the decomposition (remember that E is a 600×600 -matrix) and CLDGCR needs about half an hour (E a 2400×2400 -matrix). For solving $E\mathbf{x} = \mathbf{y}$, CDGCR needs about 0.02 seconds, CLDGCR about 0.3 seconds. For convergence, a large number of iterations is needed, so the total time for solving $E\mathbf{x} = \mathbf{y}$ all the times is very large.
The above is only one example, not every case with many subdomains will give the same results. But it is clear that when one has a geometry with many subdomains, a deflation algorithm can not be easily applied, because convergence costs too much CPU time.

There are two ways of solving this 'subdomain' problem: the first one is changing the way $E\mathbf{x} = \mathbf{y}$ is solved for $\mathbf{x}$, the second one is to decrease the number of subdomains.

As stated above, the solver that is now used to solve $E\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ is a direct solver that does not use the sparseness of E. This solver is suitable for problems with less than about fifty subdomains. For problems with more subdomains a sparse solver for unstructured matrices is needed. The numbering of the subdomains is very important when using a sparse solver, because this numbering influences the positions of the non-zeros in the matrix.

For most cases, the subdomain partitioning is made considering two things: physics and geometry. By 'considering physics' the following is meant: the geometry of a certain test case consists of different parts. For example, a glass

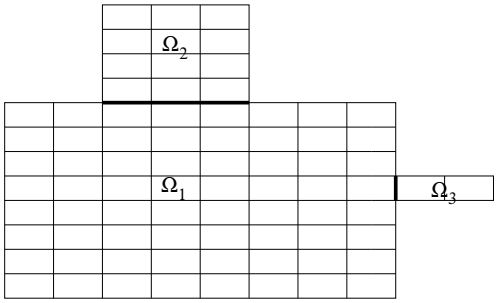

Fig. 7.15: Subdomain partitioning considering the geometry

melting furnace has a combustion chamber, a batch blanket and a glass melting part. Different equations are solved for each part. It is logical to solve the same set of equations within a subdomain. So the subdomains are chosen such that this is the case.
Taking care of the geometry when choosing the subdomains is most easily explained by a figure. Consider Figure 7.15. The subdomain partitioning is done by looking at the geometry and dividing it in subdomains in a logical way.
In practice, subdomain partitioning considering physics corresponds with subdomain partitioning considering geometry.
When there are many subdomains, it is better for deflated GCR to take subdomains together. Certain choices have to be made then. Further research has to be done to say more about this subject. When you stumble across a case with many subdomains now, it is wise to try constant deflation first, and not immediately constant linear deflation.

7.5 Accuracy of the LU decomposition of Z^TAZ

When using deflation in X-stream, the inverse of the matrix Z^TAZ has to be calculated. Therefore it is necessary that the matrix Z^TAZ is non-singular. An easy check can be done with help of the LU decomposition of Z^TAZ. It is known that when a matrix is singular, its LU decomposition is not correct. Therefore the following check is done: consider a matrix M. An LU decomposition of M can be made, resulting in a lower triangular matrix L and an upper triangular matrix U. If M is a non-singular matrix $L * U$ should be equal to M, so $L * U - M$ should be a matrix with only very small values. Below the algorithm is given that X-stream used to check whether the matrix Z^TAZ is non-singular:

Algorithm to check non-singularity Z^TAZ
1. Diff $= Z^TAZ - LU(Z^TAZ)$
2. **If** (any Diff(:,:) $>$ 1E-10) **then**
   write 'warning: deflation solver can give erroneous results!'
   End if

The factor 10^{-10} is determined empirically by looking at some test cases.
In the above algorithm, the magnitude of the numbers in the matrix Z^TAZ

is not taken into account.It is clear that when $Z^T AZ$ has numbers in the order of 10^5, the absolute errors made by the LU decomposition are larger than when the largest number in $Z^T AZ$ is of the order 10^{-5}. Therefore, it is better that some kind of scaling is done in the algorithm. We have chosen a diagonal scaling, because it scales the numbers in the original matrices $Z^T AZ$ and $LU(Z^T AZ)$ around 1. The algorithm with this scaling is given by:

Algorithm to check non-singularity $Z^T AZ$ with scaling
1. **For** i = 1, n **do**:
   Diag(i) = $\sum_{j=1}^{n} |(Z^T AZ)_{ij}|$
   End for
2. Diff = $Z^T AZ - LU(Z^T AZ)$
3. **For** i = 1, n **do**:
   Diff(i,:) = |Diff(i,:)|/Diag(i)
   End for
4. **If** (any Diff(:,:) > 1E-10) **then**
   write 'warning: deflation solver can give erroneous results!'
   End if

In step 1 a diagonal scaling is calculated. In step 2 the difference between the matrices $Z^T AZ$ and $LU(Z^T AZ)$ is calculated. In step 3 the scaling is applied: every row of the difference matrix is divided by a scaling factor.
Finally, in step 4 a warning is given when one of the elements in the scaled difference matrix is larger then 10^{-10}.

The scaling is testes for some test cases. Before the scaling was implemented, for some of these cases a lot of warnings were given. After implementing the algorithm with scaling, these warnings are no longer given for the test cases. The largest number in the matrix Diff was of the order 10^{-16}, so the faults made by the LU decomposition do not come close to the limit of 10^{-10}.

8. GCR-SIMPLE ON A COLOCATED GRID

In this chapter GCR-SIMPLE on a colocated grid will be discussed. First the algorithms GCR, SIMPLE and GCR-SIMPLE are stated in Section 8.1. In Section 8.2 diagonal scaling is described and in Section 8.3 the implementation of GCR-SIMPLE is treated.

8.1 Algorithms

In this section GCR and SIMPLE are repeated (from Chapter 4). After that they are combined in GCR-SIMPLE.

8.1.1 GCR

For a general matrix A the GCR Krylov subspace method can be used to solve the linear system $A\mathbf{x} = \mathbf{b}$. The preconditioned GCR method is given by the following algorithm, see for example Vuik, Frank & Segal [26].

Algorithm 8.1 (PGCR)
Given an $n \times n$ matrix A, an n-vector $\mathbf{b}$, a preconditioner M and an initial guess $\mathbf{x}^{(0)}$, this algorithm solves the linear system $A\mathbf{x} = \mathbf{b}$.
$\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$
For $j = 1, \ldots,$ until convergence **do**:
    Solve $M\mathbf{v}^{(j)} = \mathbf{r}^{(j-1)}$
    $\mathbf{q}^{(j)} = A\mathbf{v}^{(j)}$
    $[\mathbf{q}^{(j)}, \mathbf{v}^{(j)}] = \mathbf{orthonorm}[\mathbf{q}^{(j)}, \mathbf{v}^{(j)}, \mathbf{q}^{(i)}, \mathbf{v}^{(i)}, i < j]$
    $\gamma = (\mathbf{q}^{(j)}, \mathbf{r}^{(j-1)})$
    $\mathbf{x}^{(j)} = \mathbf{x}^{(j-1)} + \gamma\mathbf{v}^{(j)}$
    $\mathbf{r}^{(j)} = \mathbf{r}^{(j-1)} - \gamma\mathbf{q}^{(j)}$
End for
$\mathbf{x} \approx \mathbf{x}^{(j)}$

The relation between $\mathbf{q}$ and $\mathbf{v}$ ($\mathbf{q} = A\mathbf{v}$) is very important and should be satisfied for all intermediate steps. The routine **orthonorm** refers to the orthonormalization process used. It is left open in the above algorithm because there are several ways to do this, see Frank & Vuik [4]. A common used method is the Modified Gram-Schmidt algorithm:

$[\mathbf{q}, \mathbf{v}] = \mathbf{orthonorm}[\mathbf{q}, \mathbf{v}, \mathbf{q}^{(i)}, \mathbf{v}^{(i)}, i < j]$
For $i = 1, \ldots, j-1$ **do**:
$\alpha = (\mathbf{q}, \mathbf{q}^{(i)})$
$\mathbf{q} = \mathbf{q} - \alpha \mathbf{q}^{(i)}$
$\mathbf{v} = \mathbf{v} - \alpha \mathbf{v}^{(i)}$
End for
$\mathbf{q} = \mathbf{q}/\|\mathbf{q}\|_2$
$\mathbf{v} = \mathbf{v}/\|\mathbf{q}\|_2$

$\|.\|$ is the Euclidean norm (3.6).

8.1.2 SIMPLE

The algebraic system to be solved by the SIMPLE method may be denoted as

$$\begin{pmatrix} N & G \\ D & C \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix}. \tag{8.1}$$

At the k-th iteration we have

$$\mathbf{b} - A\mathbf{x}^{(k)} = \begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix} - \begin{pmatrix} N & G \\ D & C \end{pmatrix} \begin{pmatrix} \mathbf{u}^{(k)} \\ \mathbf{p}^{(k)} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_1^{(k)} \\ \mathbf{r}_2^{(k)} \end{pmatrix}. \tag{8.2}$$

In the SIMPLE algorithm below, some matrices are approximated: Q and R are approximations of N and $C - DN^{-1}G$, respectively. $\tilde{N}^{-1}$ is an easy to evaluate approximate inverse of N (in practice, one chooses $\tilde{N} = diag(N)$). For more information and background on the SIMPLE algorithm, the reader is referred to Chapter 4.

Algorithm 8.2 (SIMPLE)
Let $\mathbf{p}^{(0)}$ and $\mathbf{u}^{(0)}$ be initial estimations of the pressure and velocity field, respectively. Let ω_p and ω_u be underrelaxation parameters. This algorithm solves the non-linear system (8.1) for the stationary incompressible Navier-Stokes equations.
For $k = 0, \ldots,$ until convergence **do**:
Compute $\mathbf{r}_1^{(k)}$ and $\mathbf{r}_2^{(k)}$ from (8.2)
Solve $Q\delta\mathbf{u} = \mathbf{r}_1^{(k)}$
Solve $R\delta\mathbf{p} = \mathbf{r}_2^{(k)} - D\delta\mathbf{u}$
$\delta\mathbf{u} = \delta\mathbf{u} - \tilde{N}^{-1}G\delta\mathbf{p}$
$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \omega_u \delta\mathbf{u}$
$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} + \omega_p \delta\mathbf{p}$
End for
$\mathbf{u} \approx \mathbf{u}^{(k)}$
$\mathbf{p} \approx \mathbf{p}^{(k)}$

Experience has shown that the use of underrelaxation factors is necessary.

8.1.3 GCR-SIMPLE

In Chapter 5 information can be found on the GCR-SIMPLE method. In this subsection the most important matrices and equations are repeated for the 2D

case, after which the algorithm is given.

In GCR-SIMPLE, the system $A\mathbf{x} = \mathbf{b}$ (resulting from the discretization of the Navier-Stokes equations) is solved, with

$$A = \begin{pmatrix} N_1 & 0 & G_1 \\ 0 & N_2 & G_2 \\ D_1 & D_2 & C \end{pmatrix}, \mathbf{x} = \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{p} \end{pmatrix}, \mathbf{b} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix}. \tag{8.3}$$

Preconditioner M (in PGCR) is chosen as BP^{-1} defined by

$$B = \begin{pmatrix} I & 0 & -\tilde{N}_1^{-1}G_1 \\ 0 & I & -\tilde{N}_2^{-1}G_2 \\ 0 & 0 & I \end{pmatrix} \tag{8.4}$$

and

$$P = \begin{pmatrix} N_1 & 0 & 0 \\ 0 & N_2 & 0 \\ D_1 & D_2 & C - \sum_{i=1}^{2} D_i \tilde{N}_i^{-1} G_i \end{pmatrix}. \tag{8.5}$$

The GCR-SIMPLE method for a colocated grid arrangement can now be obtained in a similar way as for the staggered case, see Vuik & Saghir [27]. The GCR-SIMPLE method is given by the following algorithm:

Algorithm 8.3 (GCR-SIMPLE)
Consider the system $A\mathbf{x} = \mathbf{b}$ resulting from the discretization of the Navier-Stokes equations. Let the matrix A be given by (8.3) and let B and P be given by (8.4) and (8.5), respectively. Let $\mathbf{x}^{(0)}$ be an initial solution, such that $A\mathbf{x}^{(0)} \approx \mathbf{b}$. Then this algorithm solves $A\mathbf{x} = \mathbf{b}$.
$\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$
For $k = 1, \ldots,$ ngcrsimple **do**:
  Solve $P\mathbf{s} = \mathbf{r}^{(k-1)}$
  $\mathbf{v}^{(k)} = B\mathbf{s}$
  $\mathbf{q}^{(k)} = A\mathbf{v}^{(k)}$
  Orthonormalize $\mathbf{q}^{(k)}$ and $\mathbf{v}^{(k)}$
  $\beta = (\mathbf{r}^{(k-1)}, \mathbf{v}^{(k)})$
  $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \beta\mathbf{q}^{(k)}$
  $\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} - \beta\mathbf{v}^{(k)}$
End for
$\mathbf{x} \approx \mathbf{x}^{(k)}$

8.2 Diagonal scaling

It is convenient to use a diagonal scaling of the system, because this leads to a better behaviour with respect to rounding errors. To implement diagonal scaling some adaptations are made. The 'scaling' matrix is given by $D_{AB}^{-1} = diag(AB)^{-1}$ and A, $\mathbf{b}$ and P are replaced by $D_{AB}^{-1}A$, $D_{AB}^{-1}\mathbf{b}$ and $D_{AB}^{-1}P$, respectively. Now the GCR-SIMPLE method with diagonal scaling is given by the following algorithm:

Algorithm 8.4 (GCR-SIMPLE with diagonal scaling)
Consider the system $A\mathbf{x} = \mathbf{b}$ resulting from the discretization of the Navier-Stokes equations. Let the matrix A be given by (8.3) and let B and P be given by (8.4) and (8.5), respectively. Let $\mathbf{x}^{(0)}$ be an initial solution, such that $A\mathbf{x}^{(0)} \approx \mathbf{b}$. Then this algorithm solves $A\mathbf{x} = \mathbf{b}$.
$\mathbf{r}^{(0)} = D_{AB}^{-1}(\mathbf{b} - A\mathbf{x}^{(0)})$
For $k = 1, \ldots,$ ngcrsimple **do**:
  Solve $D_{AB}^{-1} M\mathbf{s} = \mathbf{r}^{(k-1)}$
  $\mathbf{v}^{(k)} = B\mathbf{q}$
  $\mathbf{q}^{(k)} = D_{AB}^{-1} A\mathbf{v}^{(k)}$
  Orthonormalize $\mathbf{q}^{(k)}$ and $\mathbf{v}^{(k)}$
  $\beta = (\mathbf{r}^{(k-1)}, \mathbf{v}^{(k)})$
  $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \beta\mathbf{q}^{(k)}$
  $\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} - \beta\mathbf{v}^{(k)}$
End for
$\mathbf{x} \approx \mathbf{x}^{(k)}$

In practice, for preconditioning an approximation $\tilde{A}$ of A is used:

$$\tilde{A} = \begin{pmatrix} Q_1 & 0 & G_1 \\ 0 & Q_2 & G_2 \\ D_1 & D_2 & C \end{pmatrix}.$$

Diagonal scaling for the colocated case is not so obvious, as we will illustrate now. Consider

$$\tilde{A}B = \begin{pmatrix} Q_1 & 0 & G_1 - Q_1\tilde{N}_1^{-1}G_1 \\ 0 & Q_2 & G_2 - Q_2\tilde{N}_2^{-1}G_2 \\ D_1 & D_2 & \hat{R} \end{pmatrix},$$

with $\hat{R} = R$ in the staggered case and $\hat{R} = R + C$ in the colocated case, R given by $R = -\sum_{i=1}^{2} D_i\tilde{N}_i^{-1}G_i$. As diagonalization matrix we should take

$$D_{AB} = \begin{pmatrix} D_1 & 0 & 0 \\ 0 & D_2 & 0 \\ 0 & 0 & D_3 \end{pmatrix}, \tag{8.6}$$

with $D_1 = diag(Q_1)$, $D_2 = diag(Q_2)$ and $D_3 = diag(R)$ for the staggered case and $D_3 = diag(C + R)$ for the colocated case. The problem for the colocated case is that the matrix C is not explicitly available. Therefore, one could choose $D_{AB} = diag(R)$, just like in the staggered case. However, when $\|C\|$ is not negligible compared to $\|R\|$, it is unknown what effect this shall have on the performance of the GCR-SIMPLE algorithm. Unfortunately, for glass furnaces, it is known that $\|C\|$ is large and an it is difficult to make an estimation of the values on the diagonal of C. Therefore, the scaling will not be very accurate.
The following algorithm is a detailed version of Algorithm 8.4. The matrices denoted by $\square$ are contributions to the diagonal scaling. Removing those matrices results in the GCR-SIMPLE algorithm without diagonal scaling.

Algorithm 8.5 (GCR-SIMPLE with diagonal scaling (detailed))
Consider the system $A\mathbf{x} = \mathbf{b}$ resulting from the discretization of the Navier-Stokes equations. Let the matrix A be given by (8.3) and let D_i be given by (8.6). Let $\mathbf{x}^{(0)}$ be an initial solution such that $A\mathbf{x}^{(0)} \approx \mathbf{b}$. Then this algorithm solves $A\mathbf{x} = \mathbf{b}$.
$\mathbf{r}_i^{(0)} = \boxed{D_i^{-1}}(\mathbf{b}_i - N_i\mathbf{u}_i^{(0)} - G_i\mathbf{p}^{(0)})$, $i = 0, \ldots, d$
$\mathbf{r}_{d+1}^{(0)} = \boxed{D_{d+1}^{-1}}(\mathbf{b}_{d+1} - \sum_{i=1}^{d} D_i\mathbf{u}_i^{(0)} - C\mathbf{p}^{(0)})$
For $k = 1, \ldots,$ ngcrsimple **do**:
Solve $\boxed{D_i^{-1}} N_i\mathbf{s}_i = \mathbf{r}_i^{(k-1)}$, $i = 0, \ldots, d$
Solve $\boxed{D_{d+1}^{-1}} R\mathbf{s}_{d+1} = \mathbf{r}_{d+1}^{(k-1)} - \boxed{D_{d+1}^{-1}} \sum_{i=1}^{d} D_i\mathbf{q}_i$
$\mathbf{v}_i^{(k)} = \mathbf{s}_i - \boxed{D_i^{-1}} G_i\mathbf{s}_{d+1}$, $i = 0, \ldots, d$
$\mathbf{v}_{d+1}^{(k)} = \mathbf{s}_{d+1}$
$\mathbf{q}_i^{(k)} = \boxed{D_i^{-1}}(N_i\mathbf{v}_i^{(k)} + G_i\mathbf{v}_{d+1}^{(k)})$, $i = 0, \ldots, d$
$\mathbf{q}_{d+1}^{(k)} = \boxed{D_{d+1}^{-1}}(\sum_{i=1}^{d} D_i\mathbf{v}_i^{(k)} + C\mathbf{v}_{d+1}^{(k)})$
Orthonormalize $\mathbf{q}^{(k)}$ and $\mathbf{v}^{(k)}$
$\beta = (\mathbf{r}^{(k-1)}, \mathbf{v}^{(k)})$
$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \beta\mathbf{q}^{(k)}$
$\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} - \beta\mathbf{v}^{(k)}$
End for
$\mathbf{x} \approx \mathbf{x}^{(k)}$

It is shown in Vuik & Saghir [27] that taking only a small number of GCR-SIMPLE steps is sufficient, and reduces the amount of work and storage introduced by the orthonormalization method.

8.3 Implementation

Before starting the Master's project, the GCR-SIMPLE method was implemented in X-stream for the largest part. However, it did not work properly yet. During the Master's project, the implementation is further improved but it is still not finished. For making a new method to work, it is always advisable to start with an elementary problem. A suitable setup is given by test case 1 (see Section 6.1), with the unit cube divided in a small number of cells. For this test case, it turns out that the pressure is smooth enough and no PWI is necessary to obtain a correct checkerboard-free solution.

Two problems to overcome in the X-stream code seem to be the treatment of the pressure and the handling of C. Since the code is based on the SIMPLE method, pressure correction is used for correcting boundary velocities and fluxes. However, the GCR-SIMPLE method does not explicitly provide pressure correction as the SIMPLE method. As stated before, $\|C\|$ is not always negligible compared to $\|R\|$ and an estimation of the diagonal of C is hard to make. These two problems could be the cause for the GCR-SIMPLE method still not working in X-stream. Further research has to be done to overcome these problems.

9. CONCLUSIONS AND RECOMMENDATIONS

9.1 Conclusions

In this thesis, deflated Krylov subspace methods and the GCR-SIMPLE method are discussed, for use of these methods in the CFD package X-stream. Deflated Krylov subspace methods can be used to accelerate additive Schwarz domain decomposition (DD) methods. To gain insight into these deflated methods, they were applied to some testproblems in MATLAB for 1D numerical experiments, as well as to some test cases in X-stream for 3D experiments. In MATLAB, PGCR and DPGCR were implemented. For the deflated method constant deflation (CD) and constant linear deflation (CLD) (a combination of constant and linear vectors) were considered. The linear vectors can be constructed in two ways: based on grid numbering (CLD-ijk) and based on grid coordinates (CLD-cartesian). Both ways were tested. The following conclusions can be drawn from the experiments in MATLAB and X-stream:

1. Concerning convergence rate, deflated (P)GCR performs better than (P)GCR. (P)GCR performs best with CLD deflation, followed by CD deflation.
2. CLDGCR-cartesian has smaller (or just as large) residuals than the other methods that are tested, therefore CLDGCR-cartesian needs less iterations than the other methods or just as much. The difference between CLDGCR-ijk and CLDGCR-cartesian can be explained by the fact that the effective condition number of $P_{CLD-ijk}A$ is larger than the effective condition number of $P_{CLD-cart}A$.
3. When solving $A\mathbf{y} = \mathbf{b}$ with A an SPD matrix and a uniform cell size in each subdomain, CLDGCR-ijk and CLDGCR-cartesian produce the same results and need the same number of iterations.
4. More subdomains means less iterations. Drawback of increasing the number of subdomains is that each iteration becomes more expensive. For every problem, there is an optimal number of subdomains such that the total wall-clock time is as low as possible.
5. Each method has an optimal number of inner iterations. When comparing the methods with their optimal number of inner iterations, CLDGCR-cartesian shows the best performance. It can make the time to converge twice as fast, compared to GCR.
6. Summarizing the conclusions above, CLDGCR-cartesian is the method that performs best compared to the other methods that are tested. However, there is one drawback: when the number of subdomains is large,

CLDGCR can be very slow, especially CLDGCR-cartesian. This is the case because of the following: In X-stream (when using deflation), $Z^T AZ\mathbf{x} = \mathbf{y}$ is solved for $\mathbf{x}$. To do this, a solver is used that does not use the sparseness of $Z^T AZ$. When there is a large number of subdomains, the computation time will increase a lot. Say there are 600 subdomains. $Z^T AZ$ then is a 600×600-matrix for CLDGCR-ijk and a 2400×2400-matrix for CLDGCR-cartesian. To solve this problem, the number of subdomains has to decrease or the way $Z^T AZ$ is solved has to change.

9.2 Recommendations

The following recommendations can be made:

1. A suitable sparse solver for unstructured matrices should be implemented to solve $Z^T AZ\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ when there are many (say more than fifty) subdomains.
2. Subdomain partitioning should be examined further. When there are many subdomains, one should decrease the number of subdomains. How this can be done best, should be investigated.
3. The GCR-SIMPLE method, which is implemented for the largest part, could be finished. Attention has to be paid to the treatment of the pressure and the matrix C.

APPENDIX A

In this appendix a derivation is given for the fact that $Q\mathbf{x}$ can be obtained solving the deflated system $PA\tilde{\mathbf{x}} = P\mathbf{b}$ for $\tilde{\mathbf{x}}$ and premultiplying the result with Q as stated in Section 3.3.
Multiplying $Q\mathbf{x}$ by A and using the fact that $PA = AQ$ gives:

$$AQ\mathbf{x} = PA\mathbf{x} = P\mathbf{b}. \tag{A.1}$$

So $\mathbf{x}$ has to satisfy $PA\mathbf{x} = P\mathbf{b}$. P is singular, so this can not be solved directly. Because $PAZ = 0$, $\mathrm{Ker}(P) = AZ$. Note that the columns of PA span $P\mathbf{b}$ so the system is compatible. This means that the number of solutions $\mathbf{x}$ is infinite. Say $\tilde{\mathbf{x}}$ is a solution of

$$PA\tilde{\mathbf{x}} = P\mathbf{b}. \tag{A.2}$$

Because $\ker(P) = AZ$ we know that $\ker(PA) = Z$, so one can write $\tilde{\mathbf{x}}$ as

$$\tilde{\mathbf{x}} = \mathbf{x} + Z\mathbf{f}, \tag{A.3}$$

with $\mathbf{f}$ an arbitrary vector. Then:

$$Q\tilde{\mathbf{x}} = Q\mathbf{x} + QZ\mathbf{f} = Q\mathbf{x}, \tag{A.4}$$

because $QZ = 0$, as stated in Section 3.3.
It is clear now that solving $PA\tilde{\mathbf{x}} = P\mathbf{b}$ for $\tilde{\mathbf{x}}$ and premultiplying the result with Q gives $Q\mathbf{x}$.

LIST OF FIGURES

BIBLIOGRAPHY

[1] Brakkee, E. Domain decomposition for the incompressible Navier-Stokes equations (PhD thesis). Delft, 1996.

[2] Choi, S.-K., Kim, S.-O., Lee, C.-H., Choi, H.-K. Use of the momentum interpolation method for flows with a large body force. Numerical Heat Transfer, Part B, Vol. 43, pp. 267-287, 2003.

[3] Ferziger, J.H., Perić, M. Computational methods for fluid dynamics. Springer, Heidelberg, 1999.

[4] Frank, J., Vuik, C. Parallel implementation of a multiblock method with approximate subdomain solution. Applied Numerical Mathematics, Vol. 30, pp. 403-423, 1999.

[5] Frank, J., Vuik, C. On the construction of deflation-based preconditioners. SIAM Journal on Scientific Computing, Vol. 23, pp. 442-462, 2001.

[6] Hackbusch, W. Multi-grid methods and applications. Springer, Berlin, 1985.

[7] Jonkers, E. Deflated Schwarz domain decomposition methods for the CFD-package X-stream (Interim Master's Thesis). TNO/TU Delft, Delft, 2005.

[8] Keyes, D.E. How scalable is domain decomposition in practice? Proceedings of the 11th Conference on Domain Decomposition Methods, pp. 286-297, Greenwich, 1998.

[9] Li, C., Vuik, C. The GCR-SIMPLE solver and the SIMPLE-type preconditioning for incompressible Navier-Stokes equations. European Congress on Computational Methods in Applied Sciences and Engineering, Jyväskylä, 2004.

[10] Nabben, R. Comparisons between multiplicative and additive schwarz iterations in domain decomposition methods. Numerical Mathematics, Vol. 95, pp. 145-162, 2003.

[11] Nicolaides, R.A. Deflation of conjugate gradients with applications to boundary value problems. SIAM Journal on Numerical Analysis, Vol. 24, No. 2, pp. 355-365, 1987.

[12] Padiy, A., Axelsson, O., Polman, B. Generalized augmented matrix preconditioning approach and its application to iterative solution of ill-conditioned algebraic systems. SIAM Journal on Matrix Analysis and Applications, Vol. 22, pp. 793-818, 2000.

[13] Patankar, S.V. Numerical heat transfer and fluid flow. McGraw-Hill, New York, 1980.

[14] Saad, Y. Iterative methods for sparse linear systems. PWS publishing company, Boston, 1996.

[15] Smith, B.F., Bjørstad, P.E., Gropp, W.D. Domain decomposition: parallel multilevel methods for elliptic partial differential equations. Cambridge University Press, Cambridge, 1996.

[16] Teigland, R. A domain decomposition strategy for simulation of industrial fluid flows. Proceedings of the 9th Conference on Domain Decomposition Methods, Ullensvang, 1998.

[17] TNO Glass Group. GTM-X user manual. Eindhoven, 2005.

[18] Twerda, A. Advanced computational methods for complex flow simulation (PhD thesis). Delft, 2000.

[19] Verkaik, J. Deflated Krylov-Schwarz domain decomposition for the incompressible Navier-Stokes equations on a colocated grid (Master's Thesis). TNO, Delft, 2003.

[20] Vermolen, F.J., Vuik, C. The influence of deflation vectors at interfaces on the deflated conjugate gradient method. Report of the Department of Applied Mathematical Analysis, Delft University of Technology, ISSN 1389-6520, Delft, 2001.

[21] Verweij, R.L. Parallel computing for furnace simulations using domain decomposition (PhD thesis). Delft, 1999.

[22] Van der Vorst, H.A. Iterative Krylov methods for large linear systems. Cambridge University Press, Cambridge, 2003.

[23] Vuik, C. Numerical methods for large algebraic systems (lecture notes). Delft, 2001.

[24] Vuik, C., Frank, J. Deflated ICCG method applied to problems with extreme contrasts in the coefficients. 16th IMACS World Congress, Lausanne, 2000.

[25] Vuik, C., Frank, J. Coarse grid acceleration of a parallel block preconditioner. Future Generation Computer Systems, Vol. 17, pp. 933-940, 2001.

[26] Vuik, C., Frank, J., Segal, A. A parallel block-preconditioned GCR method for incompressible flow problems. Future Generation Computer Systems, Vol. 18, pp. 31-40, 2001.

[27] Vuik, C., Saghir, A. The Krylov accelerated SIMPLE(R) method for incompressible flow. Report of the Department of Applied Mathematical Analysis, Delft University of Technology, ISSN 1389-6520, Delft, 2002.

[28] Vuik, C., Saghir, A., Boerstoel, G.P. The Krylov accelerated SIMPLE(R) method for flow problems in industrial furnaces. International Journal of Numerical Methods in Fluids, Vol. 33, pp. 1027-1040, 2000.

[29] Vuik, C., Segal, A., Meijerink, J.A. An efficient preconditioned CG method for the solution of a class of layered problems with extreme contrasts in the coefficients. Journal of Computational Physics, Vol. 152, pp. 385-403, 1999.

[30] Vuik, C., Segal, A., El Yaakoubi, L., Dufour, E. A comparison of various deflation vectors applied to elliptic problems with discontinuous coefficients. Applied Numerical Analysis, Vol. 41, pp. 219-233, 2002.

[31] Wesseling, P. An introduction to multigrid methods. Wiley, Chichester, 1992.

[32] Wesseling, P. Principles of computational fluid dynamics. Springer, Heidelberg, 2001.

NOTATIONS

Roman symbols

Symbol	*Description*	*Units*
a	sum of coefficients in PWI method	$[-]$
A	matrix of linear system	$[-]$
b	source term in DAS	$[-]$
B	iteration matrix	$[-]$
B	postconditioner	$[-]$
c_p	specific heat	$[m^2\ s^{-2}\ K^{-1}]$
C	linear algebraic operator in DAS	$[-]$
d	number of space dimensions	$[-]$
D	linear algebraic operator in DAS	$[-]$
e	internal energy per unit mass	$[m^2\ s^{-2}]$
e	auxiliary variable for denoting grid positions	$[-]$
e	global truncation error	$[-]$
E	matrix in deflation method	$[-]$
f	dimensionless body force	$[-]$
f	inhomogeneous term in differential equation	$[-]$
F	flux	$[-]$
g	gravitational acceleration	$[m\ s^{-2}]$
g	boundary value	$[-]$
G	computational grid	$[-]$
G	linear algebraic operator in DAS	$[-]$
h	boundary value	$[-]$
h	enthalpy per unit mass	$[m^2\ s^{-2}]$
h	cell size	$[-]$
I	identity matrix	$[-]$
I	index set	$[-]$
j	reference to a grid point	$[-]$
j	iteration number	$[-]$
k	iteration number	$[-]$
L	length scale	$[m]$
L	discrete operator	$[-]$
M	preconditioner	$[-]$
n	unit normal	$[-]$
N	nonlinear algebraic operator in DAS	$[-]$
N	remainder matrix in the splitting	$[-]$
$nblock$	total number of blocks	$[-]$
p	pressure	$[kg\ m^{-1}\ s^{-2}]$
p	dimensionless mesh Péclet number	$[-]$
P	projector in deflation method	$[-]$

P	preconditioner	$[-]$
q	source term in transport equation	$[-]$
Q	projector in deflation method	$[-]$
r	local truncation error	$[-]$
r	residual	$[-]$
R	trivial extension matrix	$[-]$
Re	Reynolds number	$[-]$
s	switch function	$[-]$
S	source term in enthalpy equation	$[-]$
t	time	$[s]$
t	dimensionless time	$[-]$
u	velocity	$[m\ s^{-1}]$
U	velocity scale	$[m\ s^{-1}]$
v	search direction in Krylov method	$[-]$
W	computing work	$[-]$
x	dimensionless coordinate	$[-]$
x	coordinate	$[m]$
Y	matrix in deflation method	$[-]$
Z	matrix in deflation method	$[-]$

Greek symbols

Symbol	*Description*	*Units*
α	reference to a coordinate	$[-]$
α	relaxation parameter	$[-]$
δ	small parameter	$[-]$
ε	inverse Péclet number	$[-]$
Γ	artificial boundary	$[-]$
Γ	effective transport coefficient	$[-]$
κ	condition number	$[-]$
λ	thermal conductivity	$[kg\ m\ s^{-3}\ K^{-1}]$
λ	eigenvalue	$[-]$
μ	dynamic viscosity	$[kg\ m^{-1}\ s^{-1}]$
ρ	density	$[kg\ m^{-3}]$
ρ	spectral radius of a matrix	$[-]$
σ	dimensionless stress tensor	$[-]$
τ	time step	$[s]$
ϕ	material property	$[-]$
φ	material property	$[-]$
ω	relaxation parameter	$[-]$
Ω	domain	$[-]$
$\partial\Omega$	domain boundary	$[-]$

Abbreviations

Abbreviation	*Description*
BC	boundary condition
BIM	basic iterative method
CD	constant deflation

CDS	central difference scheme
CFD	computational fluid dynamics
CLD	constant linear deflation
DAS	differential algebraic system
DD	domain decomposition
DPCG	deflated preconditioned conjugate gradient
DPGCR	deflated preconditioned generalized conjugate residual
FV	finite volume
GC	global communication
HDS	hybrid difference scheme
LC	local communication
LHS	left-hand side
MPI	message passing interface
PCG	preconditioned conjugate gradient
PDE	partial differential equation
PGCR	preconditioned generalized conjugate residual
PWI	pressure-weighted interpolation
RHS	right-hand side
SIMPLE	strongly implicit method for pressure-linked equations
SIMPLER	SIMPLE revised
SIP	semi-implicit procedure
SPD	symmetric positive definite
SPTDMA	space tri-diagonal matrix algorithm
SSI	SIMPLE stabilization iteration
UDS	upwind difference scheme