Deflated Krylov-Schwarz Domain Decomposition for the Incompressible Navier-Stokes Equations on a Colocated Grid

Master's Thesis

Jarno Verkaik

Delft, August 2003

Delft University of Technology | TNO TPD

Committee: Prof.dr.ir. P. Wesseling (TU Delft)
Dr.ir. C. Vuik (TU Delft)
Ir. B.D. Paarhuis (TNO TPD)
Dr. A. Twerda (TNO TPD)

Contents

Preface

This Master's thesis is written for the degree of Master of Science for the study Applied Mathematics, faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology. The graduation work is done in the unit of Numerical Mathematics of the department of Applied Mathematical Analysis. During nine months, the Master's project has been carried out at TNO TPD, division Models and Processes, department of Process Physics.

I would like to thank my direct supervisors Kees Vuik and Bart Paarhuis for their excellent supervising during the graduation work. Furthermore, I would like express my gratitude towards professor P. Wesseling, Aris Twerda, and all my colleagues at Process Physics. Last but not least, I would like to thank my family for their support.

Delft, August 2003 Jarno Verkaik

1

Introduction

Mathematical simulation of flows is important for the design, optimization and trouble shooting of glass melting furnaces. To gain insight into the glass melting process, physical experiments can be done. However, physical 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 true physics, it gives engineers in the glass industry great insight into the transport phenomena occurring in glass melting furnaces.

At Process Physics, at TNO TPD, a CFD simulation package called X-stream is developed for the glass industry. Besides solving the incompressible Navier-Stokes equations and the energy equation, several models specifically related to the process of glass melting can be solved.

The geometries of glass melting furnaces can be very complex, and generally the transport equations have to be solved with a high accuracy, resulting in large computing times. To overcome these difficulties, a domain decomposition approach can be used. The total domain is divided into subdomains, such that complex geometries can be dealt with more ease. The problem is divided into subproblems, which can be computed in parallel, and by this, computing time can be gained.

Since the incompressible Navier-Stokes equations are non-linear, solving these equations is most time-consuming. In X-stream, the Navier-Stokes equations are discretized and linearized, resulting in linear systems for the velocities and the pressure, which are solved with a domain decomposition method. Solving the system for the pressure is most time-consuming, because this system has elliptic properties.

The goal of the Master's project is to accelerate the domain decomposition method used in X-stream for solving the pressure system. To achieve this, Krylov subspace acceleration is considered combined with a new technique called deflation.

The structure of this thesis as as follows. In Chapter 2, the discretization is treated. In Chapter 3, iterative solution methods are discussed and the deflated Krylov subspace acceleration methods are described. In Chapter 4, these methods are generalized to a domain decomposition context. In Chapter 5, results are given for numerical experiments, followed in Chapter 6 by results for testcases in X-stream. Finally, in Chapter 7 conclusions and recommendations are given.

2

Finite Volume discretization

2.1 Introduction

Generally, three discretization methods can be distinguished for discretising partial differential equations (PDEs): the finite difference method, the finite element method and the finite volume method. In this chapter, the Finite Volume (FV) method will be discussed, which is in great detail treated in, for example, Patankar [16], Ferziger & Perić [4], and Wesseling [36]. In this chapter we will closely follow Wesseling [36] and entirely adopt his notation.

In Section 2.2, first the governing equations in fluid dynamics are briefly discussed. In Sections 2.3 and 2.4, respectively, the discretizations of the stationary convection-diffusion equation and the stationary incompressible Navier-Stokes equations are described. In Section 2.5, the connection with the X-stream code is given.

2.2 The governing equations in fluid dynamics

The conservation equations of mass, momentum, and energy can be found in many introduction books on fluid dynamics, for example Bird [1], Ferziger & Perić [4], or Wesseling [36].

Summation convention and Cartesian tensor notation

The conservation equations are given for a Cartesian coordinate system $(x_1, \ldots, x_d)$, where d is the number of space dimensions. Boldface Latin letters denote vectors, for example, $\boldsymbol{x} = (x_1, \ldots, x_d)$. We will use Cartesian tensor notation, in which differentiation is denoted as

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

and with a Greek subscript we refer to coordinate directions. We will adopt Einstein's summation convention: when a subscript is repeated in a term, a summation of d terms is implied. For example, the divergence of a vector field $\boldsymbol{u}$ is given by:

$$u_{\alpha,\alpha} = \sum_{\alpha=1}^{d} \frac{\partial u_\alpha}{\partial x_\alpha} \ . \tag{2.1}$$

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 $\operatorname{div} \boldsymbol{u}$.

Reynolds' transport theorem and Gauß' divergence theorem

The conservation laws of mass, momentum, and energy can be easily derived using Reynolds' transport theorem and Gauß' divergence theorem.

Let $V(t)$ be denoted as a material volume of a fluid that moves with the flow and consists permanently of the same material particles.

Theorem 2.1 (Reynolds' transport theorem). *For any material volume $V(t)$ and differentiable scalar field ϕ we have*

$$\frac{d}{dt}\int\limits_{V(t)} \phi dV = \int\limits_{V(t)} \{\frac{\partial \phi}{\partial t} + \operatorname{div}(\phi \boldsymbol{u})\} dV \ .$$

Proof. See Wesseling [36, page 10] Theorem 1.3.1. □

Gauß' divergence theorem is given by the following theorem.

Theorem 2.2 (Gauß' divergence theorem). *For any volume $V \subset \mathbb{R}^d$ with piecewise smooth closed surface S and any differentiable vector field $\boldsymbol{u}$, we have*

$$\int\limits_{V} \operatorname{div} \boldsymbol{u} dV = \int\limits_{S} \boldsymbol{u} \cdot \boldsymbol{n} dS \ ,$$

where $\boldsymbol{n}$ is the outward unit normal on S.

Proof. See Wesseling [36, page 5] Theorem 1.2.2. □

Conservation of mass

Conservation of mass means that the rate of change of mass in an arbitrary material volume $V(t)$ equals the rate of mass production in $V(t)$, *i.e.*

$$\frac{d}{dt}\int\limits_{V(t)} \rho dV = \int\limits_{V(t)} \sigma dV \ ,$$

where $\rho(t, \boldsymbol{x})$ is the density of the material particle at time t and position $\boldsymbol{x}$, and $\sigma(t, \boldsymbol{x})$ is the rate of mass production per volume. We now take $\sigma = 0$, because this is usually the case in practice. Applying Theorem 2.1, taking $\phi = \rho$, yields:

$$\int\limits_{V(t)} \{\frac{\partial \rho}{\partial t} + \operatorname{div}(\rho \boldsymbol{u})\} dV = 0 \ .$$

Since this holds for every $V(t)$, the integrand must be zero, resulting in:

$$\frac{\partial \rho}{\partial t} + (\rho u_\alpha)_{,\alpha} = 0 \ . \tag{2.2}$$

This is the mass conservation law, also known as the continuity equation.

Incompressible flow

It is clear that the velocity field $\boldsymbol{u}(t, \boldsymbol{x})$ of the flow satisfies

$$\boldsymbol{u}(t, \boldsymbol{x}) = \frac{\partial \boldsymbol{x}(t, \boldsymbol{y})}{\partial t} \ , \tag{2.3}$$

where $\boldsymbol{y}$ is an initial position at time $t = 0$. A physical property ϕ of a material particle is called a material property. The time derivative of a material property is called the total derivative, and is denoted by $D\phi/Dt$. The quantity ϕ is defined everywhere in the flow, because every material particle has some ϕ, and therefore $\phi(t, \boldsymbol{x})$ is a scalar field. We have

$$\frac{D\phi}{Dt} = \frac{\partial}{\partial t}\phi[t, \boldsymbol{x}(t, \boldsymbol{y})] = \frac{\partial \phi}{\partial t} + \frac{\partial x_\alpha(t, \boldsymbol{y})}{\partial t}\frac{\partial \phi}{\partial x_\alpha} \ ,$$

which can be written with (2.3) as

$$\frac{D\phi}{Dt} = \frac{\partial \phi}{\partial t} + u_\alpha \phi_{,\alpha} \ .$$

An incompressible flow is a flow in which the density of each material particle is constant during the motion:

$$\rho(t, \boldsymbol{x}(t, \boldsymbol{y})) = \rho(0, \boldsymbol{y}) \ .$$

When $\rho =$ constant, we get

$$\frac{D\rho}{Dt} = 0 \ , \tag{2.4}$$

and using

$$\mathrm{div}\,(\rho \boldsymbol{u}) = \rho \mathrm{div}\, \boldsymbol{u} + u_\alpha \rho_{,\alpha} \ ,$$

it follows from the continuity equation (2.2) that the flow is divergence free:

$$\mathrm{div}\, \boldsymbol{u} = 0 \ .$$

For variable density flows, *i.e.* $\rho \neq$ constant, (2.4) does not hold anymore, and we have

$$\mathrm{div}\,(\rho \boldsymbol{u}) = 0 \ ,$$

due to the continuity equation (2.2). Incompressibility is not a property of the fluid, but of the flow. It can be shown that compressibility depends only on the speed of the flow. For more details, see Wesseling [36].

Conservation of momentum

Newton's law of conservation of momentum implies that the rate of change of momentum of a material volume is equal to the total force on the volume. Two types of forces can be distinguished: body forces and surface forces. A body force acts on a material particle and is proportional to its mass. We can write

$$\text{body force} = \boldsymbol{f}^{\mathrm{b}} \rho dV(t) \ , \tag{2.5}$$

where $dV(t)$ is denoted as the volume of the material particle. A surface force works on the surface of $V(t)$ and is proportional to its area. The surface force working on the surface element $dS(t)$ of a material particle can be written as

$$\text{surface force} = \boldsymbol{f}^{\mathrm{s}} dS(t) \ . \tag{2.6}$$

Applying the law of conservation of momentum to a material volume, results in:

$$\frac{d}{dt} \int\limits_{V(t)} \rho u_\alpha dV = \int\limits_{V(t)} \rho f_\alpha^{\mathrm{b}} dV + \int\limits_{S(t)} f_\alpha^{\mathrm{s}} dS \ .$$

By applying Theorem (2.1) with $\phi = \rho u_\alpha$, we find:

$$\int\limits_{V(t)} \{\frac{\partial(\rho u_\alpha)}{\partial t} + (\rho u_\alpha u_\beta)_{,\beta}\} dV = \int\limits_{V(t)} \rho f_\alpha^{\mathrm{b}} dV + \int\limits_{S(t)} f_\alpha^{\mathrm{s}} dS \ . \tag{2.7}$$

It can be shown that there exist nine quantities $\tau_{\alpha\beta}$ such that

$$f_\alpha^{\mathrm{s}} = \tau_{\alpha\beta} n_\beta \ , \tag{2.8}$$

where $\tau_{\alpha\beta}$ is the stress tensor and $\boldsymbol{n}$ is the outward unit normal on dS. By Theorem 2.2, Equation (2.7) can be rewritten as

$$\int\limits_{V(t)} \{\frac{\partial(\rho u_\alpha)}{\partial t} + (\rho u_\alpha u_\beta)_{,\beta}\} dV = \int\limits_{V(t)} (\rho f_\alpha^{\mathrm{b}} + \tau_{\alpha\beta,\beta}) dV \ . \tag{2.9}$$

Since this holds for every $V(t)$, we must have

$$\frac{\partial(\rho u_\alpha)}{\partial t} + (\rho u_\alpha u_\beta)_{,\beta} \ = \tau_{\alpha\beta,\beta} + \rho f_\alpha^{\mathrm{b}} \ , \tag{2.10}$$

which is the momentum conservation law. The LHS of this equation is called the inertia term.

For Newtonian fluids $\tau_{\alpha\beta}$ is given by the following constitutive relation:

$$\tau_{\alpha\beta} = -p\delta_{\alpha\beta} + 2\mu(s_{\alpha\beta} - \frac{1}{3}\Delta\delta_{\alpha\beta}) \ , \tag{2.11}$$

where p is the pressure, $\delta_{\alpha\beta}$ the Kronecker delta, μ the dynamic viscosity, $s_{\alpha\beta}$ the rate-of-strain tensor, defined by

$$s_{\alpha\beta} = \frac{1}{2}(u_{\alpha,\beta} + u_{\beta,\alpha}) \ ,$$

and

$$\Delta = s_{\alpha\alpha} = \operatorname{div} \boldsymbol{u} \ .$$

The equations given by (2.10) and (2.11) are called the Navier-Stokes equations.

Conservation of energy

The first law in thermodynamics tells us that work done in a closed system plus heat added to it, equals the increase of the sum of kinetic and internal energy of the system. The kinetic energy per unit mass of a material particle is $\frac{1}{2}u_\alpha u_\alpha$, its internal energy per unit mass is denoted by e. The internal energy is a state variable, like p and ρ. For simple systems, there are at most two independent state variables needed.

Applying the first law of thermodynamics to a material volume $V(t)$ we get

$$\frac{d}{dt}\int\limits_{V(t)} \rho E dV = W + Q \ , \tag{2.12}$$

where E denotes the total energy per unit mass, and is given by

$$E = e + \frac{1}{2}u_\alpha u_\alpha \ .$$

Furthermore, W is the rate of work expended by the surroundings on the fluid $V(t)$, and Q is the rate of heat addition. The body force (2.5) does work at a rate $\boldsymbol{u}\cdot\boldsymbol{f}^b\rho dV(t)$, and the surface force (2.6) at a rate $\boldsymbol{u}\cdot\boldsymbol{f}^s\rho dS(t)$, thus W is given by

$$W = \int\limits_{V(t)} u_\alpha f^b_\alpha \rho dV + \int\limits_{S(t)} u_\alpha f^s_\alpha dS \ .$$

Using (2.8) and Theorem 2.2, this can be rewritten as

$$W = \int\limits_{V(t)} \{\rho u_\alpha f^b_\alpha + (u_\alpha \tau_{\alpha\beta})_{,\beta}\} dV \ . \tag{2.13}$$

When we assume that heat is added to each material particle at a rate q per unit of mass, and that there is a heat flux σ per unit of area through $S(t)$, we get

$$Q = \int\limits_{V(t)} \rho q dV + \int\limits_{S(t)} \sigma dS \ .$$

Using Fourier's law

$$\sigma = k\boldsymbol{n}\cdot \mathrm{grad}\, T \ ,$$

with k the thermal conductivity, and T the temperature, we get by using Theorem 2.2:

$$Q = \int\limits_{V(t)} \{\rho q + (T_{,\alpha})_{,\alpha}\} dV \ . \tag{2.14}$$

Application of Theorem 2.1 to (2.12) and substitution of (2.14) and (2.13), yields

$$\int\limits_{V(t)} \frac{\partial(\rho E)}{\partial t} + (\rho u_\alpha E)_{,\alpha}\} dV = \int\limits_{V(t)} \{(u_\alpha \tau_{\alpha\beta})_{,\beta} + (kT_{,\alpha})_{,\alpha} + \rho u_\alpha f^b_\alpha + \rho q\} dV \ .$$

Since this should hold for every $V(t)$, we have

$$\frac{\partial(\rho E)}{\partial t} + (\rho u_\alpha E)_{,\alpha} = (u_\alpha \tau_{\alpha\beta})_{,\beta} + (kT_{,\alpha})_{,\alpha} + \rho u_\alpha f^b_\alpha + \rho q \ .$$

This equation is called the energy equation.

State equations

We have obtained a total of five conservation equations: one for mass, one for momentum in each coordinate direction, and one for energy. However, there are seven unknowns: ρ, u_α ($\alpha = 1, 2, 3$), p, T and e. Therefore, the system of equations need to be completed by two additional equations, *i.e.* by two equations of state, see Wesseling [36] for more details.

General form of the conservation equations

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

$$\frac{\partial(\rho\varphi)}{\partial t} + F_{\alpha,\alpha} = S_\varphi \ , \tag{2.15}$$

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

$$F_\alpha = F_\alpha^{\text{con}} + F_\alpha^{\text{dif}} = \rho u_\alpha \varphi - \Gamma_\varphi \varphi_{,\alpha} \ , \tag{2.16}$$

where Γ_ϕ is the effective transport coefficient. For example, for the continuity equation $\phi = 1$, $\Gamma_\phi = 0$ and $S_\phi = 0$. Equation (2.15) with (2.16) is called the convection-diffusion equation.

2.3 Discretization of the stationary convection-diffusion equation

2.3.1 Problem description

Taking $\rho = 1$, dropping the subscript of Γ_φ, and renaming S_ϕ by q, the stationary form of the convection-diffusion equation given by equations (2.15) and (2.16), can be written as

$$(u_\alpha \varphi)_{,\alpha} - (\Gamma \varphi_{,\alpha})_{,\alpha} = q \ , \quad \boldsymbol{x} \in \Omega \subset \mathbb{R}^d \ .$$

We assume that this equation is linear, with the quantity φ to be the only unknown.

The dimensionless form of the above equation is (see Wesseling [36, page 111]):

$$\mathcal{L}\varphi = q \ , \quad \mathcal{L}\varphi \equiv (u_\alpha \varphi)_{,\alpha} - (\varepsilon \varphi_{,\alpha})_{,\alpha} \ , \tag{2.17}$$

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

$$\text{Pe} = \frac{UL}{\Gamma} \ ,$$

with L and U typical length and velocity scales. For $\text{Pe} \ll 1$ we have dominating convection, for $\text{Pe} \gg 1$ diffusion dominates.

The convection-diffusion equation can be classified as a parabolic equation, just like the Navier-Stokes equations (2.10). However for $0 < \Gamma \ll 1$, or $\text{Pe} \gg 1$, hyperbolic aspects are dominant. A kind of mixture of parabolic and hyperbolic behavior is typical for the convection-diffusion equation.

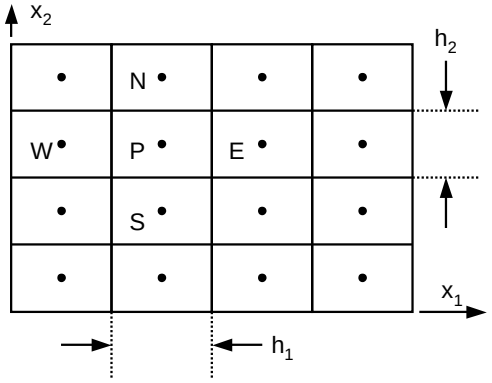

Figure 2.1: A cell-centered grid. (• grid points; – finite volume boundaries.)

Boundary conditions

One has to specify suitable boundary conditions (BCs) to assure that the problem is well-posed. Firstly, the BCs have to be chosen such that the problem has an unique solution. For a second order equation, such as the general convection diffusion equation, this means that we have to prescribe exactly one BC at each boundary of the domain Ω. Secondly, the problem is well-posed if small perturbations in the data do not cause large changes in the solution. Suitable boundary conditions for $\varepsilon \ll 1$ are given by, see Wesseling [36],

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

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

2.3.2 Interpolation practices

Approximating the diffusive flux can be done in a straightforward manner. For approximating the convective flux various methods exist. This can be done, for example, by applying a central, upwind, or hybrid scheme, or by defect correction. More accurate schemes could also used as well, see for a survey Ferziger & Perić [4, Section 4.4–4.5], and Wesseling [36, Section 4.5–4.8].

We restrict ourselves to the treatment of central and upwind discretization, and to the treatment of defect correction. First the computational grid is defined.

The computational grid

For sake of simplicity, we restrict ourselves to the two-dimensional case. Generalization to three dimensions can be done in a straightforward way. We will use a cell-centered uniform grid, see Figure 2.1. The rectangular domain $\Omega = L_1 \times L_2$ is subdivided in rectangular cells

of size $h_1 \times h_2$. The computing grid G is the set of cell-centers:

$$G = \{\boldsymbol{x} \in \Omega : \boldsymbol{x} = \boldsymbol{x_j} \ , \ \ \boldsymbol{j} = (j_1, j_2) \ , \ \ j_\alpha = 1, 2, \ldots, m_\alpha \ , \ \ m_\alpha = \frac{L_\alpha}{h_\alpha}\} \ ,$$

with $\boldsymbol{x_j}$ defined by

$$\boldsymbol{x_j} = (x_{\boldsymbol{j}}^1, x_{\boldsymbol{j}}^2) \ , \ \ x_{\boldsymbol{j}}^1 = (j_1 - \tfrac{1}{2})h_1 \ , \ \ x_{\boldsymbol{j}}^2 = (j_2 - \tfrac{1}{2})h_2 \ .$$

The cell with center $\boldsymbol{x_j}$ is denoted by $\Omega_{\boldsymbol{j}}$. Define:

$$\boldsymbol{e}_1 \equiv (\tfrac{1}{2}, 0) \ , \ \ \boldsymbol{e}_2 \equiv (0, \tfrac{1}{2}) \ ,$$

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

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

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

Central and upwind discretization

Consider an uniform cell-centered grid, as depicted in Figure 2.1. Approximating the convective flux with the central difference scheme (CDS), results in

$$(u\varphi)_{\mathrm{cds},\boldsymbol{j}+\boldsymbol{e}_1} \approx \frac{1}{2} u_{\boldsymbol{j}+\boldsymbol{e}_1}(\varphi_{\boldsymbol{j}} + \varphi_{\boldsymbol{j}+2\boldsymbol{e}_1}) \ .$$

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

$$(u\varphi)_{\mathrm{uds},\boldsymbol{j}+\boldsymbol{e}_1} \approx \begin{cases} u_{\boldsymbol{j}+\boldsymbol{e}_1}\varphi_{\boldsymbol{j}} & \text{if } \ u_{\boldsymbol{j}+\boldsymbol{e}_1} > 0 \ , \\ -u_{\boldsymbol{j}+\boldsymbol{e}_1}\varphi_{\boldsymbol{j}+2\boldsymbol{e}_1} & \text{if } \ u_{\boldsymbol{j}+\boldsymbol{e}_1} \leq 0 \ . \end{cases}$$

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

Spurious wiggles

Writing the convection-diffusion equation in non-conservation form, one can formulate the so-called maximum principle, which can give us a-priori information. This principle tells us if the exact solution has a local maximum or minimum. If this is true, wiggles in the numerical solution must be regarded as numerical artifacts.

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 that gives us the conditions for which numerical wiggles may occur.

One can verify that for the convection-diffusion equation with a constant velocity field, the UDS scheme is positive for all Péclet numbers and satisfies the discrete maximum principle, so that this scheme does not introduce wiggles. On the other hand, one can verify that the central scheme introduces wiggles for

$$p_{\boldsymbol{j}+\boldsymbol{e}_\alpha} \equiv \frac{|u_{\boldsymbol{j}+\boldsymbol{e}_\alpha}| h_\alpha}{\varepsilon} \geq 2 \ , \tag{2.18}$$

where p is called the dimensionless mesh Péclet number. See Wesseling [36] for more details.

Defect correction

Defect correction, also known as deferred correction, is an iterative method to improve the accuracy of a lower order discretization, without having to solve for a higher order discretization. More details on defect correction can be found in Wesseling [36, Section 4.6]. Let the system of equations corresponding to a lower and a higher order discretization for the stationary convection-diffusion equation be denoted by, respectively,

$$\bar{L}_h \bar{\varphi}_h = \bar{q}_h \ , \quad L_h \psi_h = q_h \ .$$

Defect correction is given by

$$\begin{aligned} &\bar{L}_h \varphi_h^{(0)} = \bar{q}_h \ , \\ &\bar{L}_h \varphi_h^{(k)} = q_h + \beta(\bar{L}_h - L_h)\varphi_h^{(k-1)}, \ k = 1, \ldots, \ , \end{aligned}$$

where $0 \leq \beta \leq 1$ is a blending factor. If $\bar{L}_h$ is first order accurate (for example UDS) and L_h is a second order scheme (for example CDS), one can prove for $\beta = 1$ that $\varphi_h^{(1)}$ is of second order accuracy.

Compared to a second order scheme, defect correction has the advantage that the resulting system has better properties when solved with an iterative solver.

2.3.3 Spatial discretization

Discretization for the interior

We will now integrate Equation (2.17) over a cell $\Omega_{\boldsymbol{j}}$. Integrating the LHS of (2.17) and using Gauß' divergence theorem (Theorem 2.2), yields

$$\begin{aligned} L_h \varphi_{\boldsymbol{j}} \equiv \int_{\Omega_j} \mathcal{L}\varphi d\Omega &= \left[\int_{\boldsymbol{x}_{\boldsymbol{j}+\boldsymbol{e}_1+\boldsymbol{e}_2}}^{\boldsymbol{x}_{\boldsymbol{j}+\boldsymbol{e}_1+\boldsymbol{e}_2}} - \int_{\boldsymbol{x}_{\boldsymbol{j}-\boldsymbol{e}_1+\boldsymbol{e}_2}}^{\boldsymbol{x}_{\boldsymbol{j}-\boldsymbol{e}_1+\boldsymbol{e}_2}} \right] (u^1\varphi - \varepsilon\varphi_{,1}) dx_2 \\ &\quad + \left[\int_{\boldsymbol{x}_{\boldsymbol{j}-\boldsymbol{e}_1+\boldsymbol{e}_2}}^{\boldsymbol{x}_{\boldsymbol{j}+\boldsymbol{e}_1+\boldsymbol{e}_2}} - \int_{\boldsymbol{x}_{\boldsymbol{j}-\boldsymbol{e}_1-\boldsymbol{e}_2}}^{\boldsymbol{x}_{\boldsymbol{j}+\boldsymbol{e}_1-\boldsymbol{e}_2}} \right] (u^2\varphi - \varepsilon\varphi_{,2}) dx_1 \\ &= F^1\big|_{\boldsymbol{j}-\boldsymbol{e}_1}^{\boldsymbol{j}+\boldsymbol{e}_1} + F^2\big|_{\boldsymbol{j}-\boldsymbol{e}_2}^{\boldsymbol{j}+\boldsymbol{e}_2} \ , \end{aligned}$$

where L_h is defined as a discrete operator, and F^α is the (numerical) flux. The surface integrals in the above equation can be approximated by the midpoint rule. When the derivative is approximated by central differences, the diffusive flux, for example at the 'east' side, is given by

$$\int_{\boldsymbol{x}_{\boldsymbol{j}+\boldsymbol{e}_1-\boldsymbol{e}_2}}^{\boldsymbol{x}_{\boldsymbol{j}+\boldsymbol{e}_1+\boldsymbol{e}_2}} (-\varepsilon\varphi_{,1}) dx_2 \approx -\varepsilon(\varphi_{\boldsymbol{j}+2\boldsymbol{e}_1} - \varphi_{\boldsymbol{j}}) \frac{h_2}{h_1} \ .$$

The convective flux can be approximated by an interpolation scheme, as described in Subsection 2.3.2.

Integrating the source term in the RHS of Equation (2.17) yields

$$\int_{\Omega_j} q d\Omega \approx \hat{q}_{\boldsymbol{j}} \equiv h_1 h_2 q(\boldsymbol{x}_{\boldsymbol{j}}) \ .$$

Discretization of the boundary conditions

Consider the case of a Dirichlet BC at the inflow, and Neumann BC at the outflow.

Let $\Omega_{\boldsymbol{j}}$ be a cell for which $(j_1 = 1, j_2)$, so that the 'west' face is part of the inflow boundary. The Dirichlet BC at the inflow is given by

$$\varphi(\boldsymbol{x}) = a \ , \quad \boldsymbol{x} \in \partial\Omega_{\text{in}} \ .$$

The convective flux $(u^1\varphi)_{\boldsymbol{j}-\boldsymbol{e}_1}$ could be approximated in a straightforward manner:

$$(u^1\varphi)_{\boldsymbol{j}-\boldsymbol{e_1}} \approx u^1_{\boldsymbol{j}-\boldsymbol{e_1}} a \ ,$$

and the diffusive flux $(\varepsilon\varphi_{,1})_{\boldsymbol{j}-\boldsymbol{e}_1}$ by a one-sided approximation:

$$(\varepsilon\varphi_{,1})_{\boldsymbol{j}-\boldsymbol{e}_1} \approx 2\varepsilon_{\boldsymbol{j}-\boldsymbol{e_1}} \frac{\varphi_{\boldsymbol{j}} - a}{h_1} \ .$$

Consider $\Omega_{\boldsymbol{j}}$ $(j_1, j_2 = m_2)$, so that the 'east' face is part of the outflow boundary. The Neumann BC at the outflow is given by:

$$\frac{\partial\varphi(\boldsymbol{x})}{\partial x_1} = b \ , \quad \boldsymbol{x} \in \partial\Omega_{\text{out}} \ .$$

Then the diffusive flux $(\varepsilon\varphi_{,1})_{\boldsymbol{j}+\boldsymbol{e}_1}$ can be simply approximated by:

$$(\varepsilon\varphi_{,1})_{\boldsymbol{j}+\boldsymbol{e}_1} \approx \varepsilon_{\boldsymbol{j}+\boldsymbol{e_1}} b \ .$$

When a UDS scheme is used for the diffusive flux $(\varepsilon\varphi_{,1})_{\boldsymbol{j}+\boldsymbol{e}_1}$, $u^1_{\boldsymbol{j}+\boldsymbol{e_1}} > 0$ is assumed, then

$$(u^1\varphi)_{\boldsymbol{j}+\boldsymbol{e_1}} \approx u^1_{\boldsymbol{j}+\boldsymbol{e_1}} \varphi_{\boldsymbol{j}} \ .$$

Using a CDS scheme, we can approximate $\varphi_{\boldsymbol{j}+2\boldsymbol{e_1}}$ with linear extrapolation, using the Neumann BC:

$$\varphi_{\boldsymbol{j}+2\boldsymbol{e_1}} \approx \varphi_{\boldsymbol{j}} + bh_1 \ .$$

In general, various methods can be considered for treating the BCs, compare Ferziger & Perić [4] to Wesseling [36].

2.4 The stationary incompressible Navier-Stokes equations

In this section, the discretization of the stationary incompressible Navier-Stokes equations is discussed. In Subsection 2.4.1, first a problem description is given, followed by the discretization on a colocated grid in Subsection 2.4.2.

2.4.1 Problem description

We restrict ourselves to the stationary case. The dimensionless form of the continuity equation (2.2) is given by:

$$u^\alpha_{,\alpha} = 0 \ . \tag{2.19}$$

The dimensionless form of the Navier-Stokes equations (2.10) and (2.11) (see Wesseling [36, page 16]) reads:

$$(\rho u^\alpha u^\beta)_{,\beta} = -p_{,\alpha} + \sigma^{\alpha\beta}_{,\beta} + f^\alpha, \quad \sigma^{\alpha\beta} = \mathrm{Re}^{-1}(u^\alpha_{,\beta} + u^\beta_{,\alpha}) \ , \tag{2.20}$$

where $\boldsymbol{f}$ is a body force, $\sigma^{\alpha\beta}$ the deviatoric stress tensor, and Re the dimensionless Reynolds number, defined by

$$\mathrm{Re} = \frac{\rho_0 U L}{\mu} \ ,$$

where ρ_0 is a suitable value for the density, U respectively L are typical length and velocity scales, and μ is the dynamic viscosity.

Note that the Navier-Stokes equations are non-linear, by the LHS of Equation (2.20).

No-slip condition

At a solid surface we have the no-slip condition

$$u^\alpha(\boldsymbol{x}) = v^\alpha(\boldsymbol{x}) \ , \quad \boldsymbol{x} \in \Omega_{\mathrm{sw}} \ ,$$

with $v^\alpha(\boldsymbol{x})$ the local wall velocity.

Inflow conditions

Since the momentum equations resemble convection-diffusion equations for the velocities, we use the same inflow BCs as in Subsection 2.3.1, *i.e.* Dirichlet conditions.

Outflow conditions

At the outflow boundary, apart from the pressure, there is usually not enough physical information available on which a sufficient number of BCs can be based. Because of the resemblance of Equation (2.20) to the convection-diffusion equation, it can be shown, by applying singular perturbation analysis that the 'wrong' information generated by the artificial outflow BC propagates upstream by a distance of $\mathcal{O}(\mathrm{Re}^{-1})$. So for highly turbulent flow, $\mathrm{Re} \gg 1$, the outflow condition does not influence the the solution significantly. However, for laminar low Reynolds flow this is not the case. To obtain a well-posed problem, it is advisable to choose a (homogeneous) Neumann outflow BC for the stationary case, see Wesseling [36].

2.4.2 Discretization on a colocated grid

Colocated and staggered grids

There are two ways to arrange the unknowns on the grid: colocated arrangement and staggered arrangement, see Figure 2.2. When all discrete unknowns are located in the cell-centers, the grid is called a colocated grid (colocate = to locate together). When the pressure is located in the cell-centers and the velocity components are located at the cell face centers, the grid is called staggered.

We will restrict ourselves to a colocated grid and to incompressible flow with constant density $\rho = 1$.

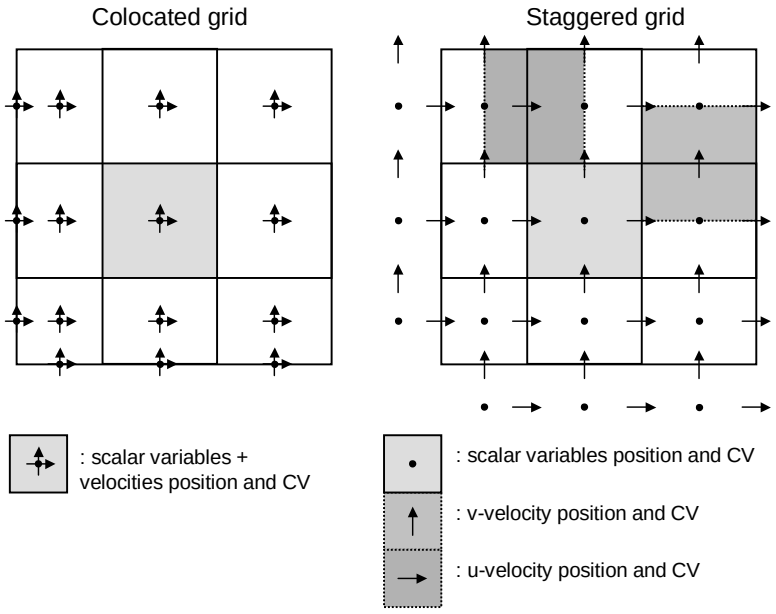

Figure 2.2: A two-dimensional domain with a colocated grid arrangement (left) and a staggered grid arrangement (right). (CV: control volume or cell.)

Discretization of the continuity equation

FV discretization of Equation (2.19) gives

$$\int_{\Omega_j} u^{\alpha}_{,\alpha} d\Omega \approx h_2 u^1|^{\boldsymbol{j}+\boldsymbol{e}_1}_{\boldsymbol{j}-\boldsymbol{e}_1} + h_1 u^2|^{\boldsymbol{j}+\boldsymbol{e}_2}_{\boldsymbol{j}-\boldsymbol{e}_2} = 0 \ . \tag{2.21}$$

Since the velocity components are situated in the cell centers, the cell face values in the above equation need to be interpolated, *e.g.*, by applying a CDS scheme:

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

Discretization of the momentum equations

For simplicity we take $\boldsymbol{f} = 0$ in (2.20), so we have to discretize

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

FV discretization yields

$$\int_{\Omega_j} \{\bar{F}^{\alpha\beta}_{,\beta} + p_{,\alpha}\} d\Omega \approx h_2 \bar{F}^{\alpha 1}|^{\boldsymbol{j}+\boldsymbol{e}_1}_{\boldsymbol{j}-\boldsymbol{e}_1} + h_1 \bar{F}^{\alpha 2}|^{\boldsymbol{j}+\boldsymbol{e}_2}_{\boldsymbol{j}-\boldsymbol{e}_2} + h_\gamma p|^{\boldsymbol{j}+\boldsymbol{e}_\alpha}_{\boldsymbol{j}-\boldsymbol{e}_\alpha} \ , \tag{2.22}$$

with $\gamma \neq \alpha$. Just as for the discretization of the continuity equation, the cell face values have to be interpolated between cell center values. For the pressure this is done as follows:

$$p_{\boldsymbol{j}+\boldsymbol{e}_\alpha} \approx \frac{1}{2}(p_{\boldsymbol{j}} + p_{\boldsymbol{j}+2\boldsymbol{e}_\alpha}) \ .$$

When $\alpha = \beta$, the deviatoric viscous stress is approximated by

$$(\sigma^{\alpha\alpha})_{\boldsymbol{j}+\boldsymbol{e}_1} = (2\mathrm{Re}^{-1}u^{\alpha}_{,\alpha})_{\boldsymbol{j}+\boldsymbol{e}_1} \approx 2\mathrm{Re}^{-1}_{\boldsymbol{j}+\boldsymbol{e}_1}(u^{\alpha}_{\boldsymbol{j}+2\boldsymbol{e}_1} - u^{\alpha}_{\boldsymbol{j}})/h_\alpha \ .$$

For the inertia terms one can use the CDS scheme:

$$(u^\alpha u^\beta)_{\boldsymbol{j}+\boldsymbol{e}_\beta} \approx \frac{1}{2}[(u^\alpha u^\beta)_{\boldsymbol{j}} + (u^\alpha u^\beta)_{\boldsymbol{j}+2\boldsymbol{e}_\beta}] \ .$$

Spurious checkerboard pattern

Using the CDS scheme to approximate the terms in Equations (2.21) and (2.22), causes a spurious checkerboard pattern. Assuming that Re = constant and neglecting the boundary conditions, one can show that

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

is a solution of the discretised Navier-Stokes equations (2.22). This shows that if $\mathrm{Re} \ll 1$, the checkerboard pattern is damped slowly for the velocity, but not at all for the pressure. A way to avoid checkerboard patterns is to use a staggered grid instead of a colocated grid. If we do not want this, another option is to use the pressure-weighted interpolation method, which will be discussed next.

Pressure weighted interpolation method

With the pressure weighted interpolation (PWI) method (or Rhie & Chow interpolation after its inventors), the cell face velocities are approximated as follows:

$$u^\alpha_{\boldsymbol{j}+\boldsymbol{e}_\alpha} = \frac{1}{2}(u^\alpha_{\boldsymbol{j}} + u^\alpha_{\boldsymbol{j}+2\boldsymbol{e}_\alpha}) + (\frac{h_\beta}{4a^\alpha}\Delta^\alpha p)|^{\boldsymbol{j}+2\boldsymbol{e}_\alpha}_{\boldsymbol{j}} \quad \text{(no summation)} \ , \tag{2.23}$$

where $\Delta^\alpha p_{\boldsymbol{j}} = p_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} - 2p_{\boldsymbol{j}} + p_{\boldsymbol{j}-2\boldsymbol{e}_\alpha}$, $\beta \neq \alpha$. See Appendix A.1 for a derivation. This discretization method is $\mathcal{O}(h_1^2+h_2^2)$ accurate. The second term in the RHS can be interpreted as a regularizing term that excludes spurious patterns.

Substitution of (2.23) in (2.21) results in the following discretization of $u^\alpha_{,\alpha}$ with the PWI method:

$$\begin{aligned} &h_2 u^1|^{\boldsymbol{j}+2\boldsymbol{e}_1}_{\boldsymbol{j}-2\boldsymbol{e}_1} + h_1 u^2|^{\boldsymbol{j}+2\boldsymbol{e}_2}_{\boldsymbol{j}-2\boldsymbol{e}_2} \\ &+ h_2^2[(\frac{1}{2a^1}\Delta^1 p)_{\boldsymbol{j}+2\boldsymbol{e}_1} - (\frac{1}{a^1}\Delta^1 p)_{\boldsymbol{j}} + (\frac{1}{2a^1}\Delta^1 p)_{\boldsymbol{j}-2\boldsymbol{e}_1}] \\ &+ h_1^2[(\frac{1}{2a^2}\Delta^2 p)_{\boldsymbol{j}+2\boldsymbol{e}_2} - (\frac{1}{a^2}\Delta^2 p)_{\boldsymbol{j}} + (\frac{1}{2a^2}\Delta^2 p)_{\boldsymbol{j}-2\boldsymbol{e}_2}] = 0 \ . \end{aligned} \tag{2.24}$$

See Appendix A.2 for a derivation of this equation.

Boundary conditions and pressure

A disadvantage of the PWI method is that it requires some further specification of conditions at boundaries, beyond what is given for the differential equations. Let $(j = 1, j_2)$, so that the 'west' face of Ω_j is part of the boundary. When the PWI method (2.23) is applied in (2.21) and (2.22), $p_{\boldsymbol{j}-2\boldsymbol{e}_1}$ is needed, corresponding to a grid point outside the grid G. This value is approximated by extrapolation from the interior:

$$p_{\boldsymbol{j}-2\boldsymbol{e}_1} = 2p_{\boldsymbol{j}} - p_{\boldsymbol{j}+2\boldsymbol{e}_1} \ ,$$

which is artificial since the differential equations are not accompanied by a boundary condition for the pressure.

When the pressure distribution is required at the boundaries, this can also be obtained by extrapolation from the interior, for example by

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

For more details we refer to Wesseling [36].

Summary of equations

After spatial discretization of the continuity equation and the Navier-Stokes equations, the following system of ordinary differential equations is obtained,

$$\begin{aligned} N(\boldsymbol{u}) + G\boldsymbol{p} &= \boldsymbol{b}_1 \ , \\ D\boldsymbol{u} + C\boldsymbol{p} &= \boldsymbol{b}_2 \ , \end{aligned} \tag{2.25}$$

where N is a non-linear algebraic operator arising from the discretization of the inertia and viscous terms of the Navier-Stokes equations, G is a linear algebraic operator representing the discretization of the pressure gradient of the Navier-Stokes equations, D and C are linear algebraic operators corresponding to the velocity terms and pressure terms, respectively, in the PWI discretization of the continuity equation. The terms $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are source terms, arising from the boundary conditions and body forces. The system (2.25) contains both differential and algebraic systems and is therefore called a differential-algebraic system (DAS).

2.5 Connection with the X-stream code

With the X-stream code, the basic governing equations, as discussed in Section 2.2, can be solved both in the glass melting space as in the combustion space. In X-stream, several additional models are of great importance, which were not discussed in this chapter, for example: combustion, turbulence, radiation, batch, electrical boosting, foam, bubbling, stirring, etc. Most of these models are specifically related to the process of glass melting. Details on this kind of models can be found in Post [17], Twerda [20] or Verweij [22].

In the X-stream code, the FV method is used on a colocated grid. The grid is structured and boundary-fitted, see for details Ferziger & Perić [4] or Wesseling [36]. Defect correction is applied and blending can be done with several schemes: UDS scheme, CDS scheme,

and higher order schemes (LDS, QUICK, TVD scheme using UMIST, MUSCL and SMART flux delimiters). Both the instationary as the stationary (non-dimensionless) incompressible Navier-Stokes equations can be solved. The PWI method is used to eliminate the checker-board modes for the pressure.

3

Iterative solution methods

3.1 Introduction

The algebraic systems of equations arising from FV discretization are generally very large and sparse, because many grid points are required for sufficient accuracy. Therefore, iterative methods are more efficient and demand far less storage capacity than direct methods, especially for the three-dimensional case.

We restrict ourselves to iterative methods. In Section 3.2, iterative methods for solving linear equations are discussed in fair detail. In Section 3.3, iterative solution methods are discussed for solving the non-linear system of the stationary incompressible Navier-Stokes equations. Finally, in Section 3.4, the connection with the X-stream code is given.

3.2 Methods for solving linear equation systems

Various methods exist for solving linear algebraic systems, each having their own advantages and disadvantages. In this section, first the principle of the basic iterative method is discussed in Subsection 3.2.1. Next, an efficient basic iterative method, called the SIP method, is given in Subsection 3.2.2. In Subsection 3.2.3, a class of efficient solution methods, called Krylov subspace methods, is discussed. These methods can be combined with the deflation technique. This is discussed in Subsection 3.2.4.

3.2.1 Basic iterative solution methods

The basic idea

Consider the linear algebraic $n \times n$ system

$$A\boldsymbol{y} = \boldsymbol{b} \ . \tag{3.1}$$

The basic idea behind iterative methods is to replace this system by some nearby system that is easier to solve. Since $\boldsymbol{b} - A\boldsymbol{y}^{(k)}$ is small if $\boldsymbol{y}^{(k)}$ is close to $\boldsymbol{y}$, the following iteration process is motivated

$$\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + M^{-1}(\boldsymbol{b} - A\boldsymbol{y}^{(k-1)}), \quad k = 1, 2, \ldots, \ , \tag{3.2}$$

which is called a basic iterative method (BIM). The formula $A = M - N$ is called a splitting of A. Iteration (3.2) can also be combined with underrelaxation:

$$M\delta\boldsymbol{y} = \alpha(\boldsymbol{b} - A\boldsymbol{y}^{(k-1)}) \ , \quad \boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + \delta\boldsymbol{y} \ , \tag{3.3}$$

with α a given relaxation parameter.

Preconditioning

Iteration (3.2) can be written into the following form:

$$\boldsymbol{y}^{(k)} = M^{-1}N\boldsymbol{y}^{(k-1)} + M^{-1}\boldsymbol{b} \ ,$$

where the matrix N is given by $N = M - A$. This iteration can be viewed as a technique for solving the system

$$(I - M^{-1}N)\boldsymbol{y} = M^{-1}\boldsymbol{b} \ ,$$

which equals

$$M^{-1}A\boldsymbol{y} = M^{-1}\boldsymbol{b} \ . \tag{3.4}$$

This system, which has the same solution as the original system, is called a left-preconditioned system and the matrix M is called the left-preconditioning matrix or simply left-preconditioner.

Another interpretation of iteration (3.2) is the following. When (3.2) is multiplied by M and $\boldsymbol{y}^{(k)} = M^{-1}\boldsymbol{x}^{(k)}$ is defined, we get

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + \boldsymbol{b} - AM^{-1}\boldsymbol{x}^{(k-1)} \ .$$

This can be viewed as a technique for solving the system

$$AM^{-1}\hat{\boldsymbol{y}} = \boldsymbol{b} \ , \quad \boldsymbol{y} = M^{-1}\hat{\boldsymbol{y}} \ , \tag{3.5}$$

which is called a right-preconditioned system, and in this context the matrix M is called a right-preconditioner.

In general, a preconditioner M transforms a linear system into an equivalent system, for which the transformed coefficient matrix has a more favorable eigenvalue spectrum. The requirements on the matrix M are the following:

- M is a good approximation to A in some sense,
- the eigenvalues of the preconditioned matrix are to be clustered around 1,
- the costs of the construction of M are not prohibitive,
- a system involving M is much easier to solve than the original system.

More details on preconditioning can be found in, for example, Saad [18] or van der Vorst [34].

From now on we consider the left-preconditioned system (3.4), and we will denote this system by

$$\tilde{A}\boldsymbol{y} = \tilde{b} \ ,$$

where $\tilde{\boldsymbol{b}} \equiv M^{-1}\boldsymbol{b}$ and $\tilde{A} \equiv M^{-1}A$.

Some convergence aspects

Rewriting (3.2), results into the preconditioned Richardson's iteration:

$$\boldsymbol{y}^{(k)} = \tilde{\boldsymbol{b}} + (I - \tilde{A})\boldsymbol{y}^{(k-1)} = \boldsymbol{y}^{(k-1)} + \tilde{\boldsymbol{r}}^{(k-1)} \ , \tag{3.6}$$

with the residual $\tilde{\boldsymbol{r}}^{(k)} \equiv \tilde{\boldsymbol{b}} - \tilde{A}\boldsymbol{y}^{(k)}$. Multiplication by -1 and adding $\boldsymbol{y}$, results in

$$\boldsymbol{y} - \boldsymbol{y}^{(k)} = \boldsymbol{y} - \boldsymbol{y}^{(k-1)} - \tilde{\boldsymbol{r}}^{(k-1)} \ .$$

With the global error defined as $\boldsymbol{e}^{(k)} = \boldsymbol{y} - \boldsymbol{y}^{(k)}$, and using the relation $\tilde{\boldsymbol{r}}^{(k)} = \tilde{A}\boldsymbol{e}^{(k)}$, this can be written as

$$\boldsymbol{e}^{(k)} = (I - \tilde{A})\boldsymbol{e}^{(k-1)} \ ,$$

which is equal to

$$\boldsymbol{e}^{(k)} = (I - \tilde{A})^k \boldsymbol{e}^{(0)} \ .$$

From this, we see that

$$\|\boldsymbol{e}^{(k)}\| \leq \|B^k\| \ \|\boldsymbol{e}^{(0)}\| \ ,$$

where $B \equiv I - \tilde{A}$ and $\|\cdot\|$ represents some norm. This shows that convergence in governed by the matrix B, which is therefore called the iteration matrix. Now define the spectral radius of a matrix A by

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

It can be shown (see for example Saad [18]) that $\rho(B)$ satisfies

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

and therefore we have convergence if and only if

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

If the splitting $A = M - N$ is a regular splitting, *i.e.*, $M^{-1} \geq 0$, $N \geq 0$, and A belongs to the class of so-called M-matrices, then it can be proven that a BIM converges. A non-singular $n \times n$ matrix A has the M-matrix property if $A^{-1} \geq 0$ and $a_{i,j} \leq 0$ for all $i \neq j$. In practice, the M-matrix property holds for important classes of discretized PDEs.

Starting vectors

Iterative solution methods, such as the BIM (3.2), need to be provided with a suitable starting vector $\boldsymbol{y}^{(0)}$. If no further information is available, one always starts with $\boldsymbol{y}^{(0)} = \boldsymbol{0}$, where $\boldsymbol{0}$ denotes a vector of appropriate size containing all zeros.

In CFD codes, generally a large number of iteration processes is nested. For example, the solution to a non-linear problem is in general approximated by the solution to a number of linear systems. In such a problem, usually the most recent iterate of the outer iteration is used as a starting vector for the inner iteration solution method.

Another example is when a multigrid method provides a starting vector. A solution computed on a coarser grid could be used as a starting vector for the solution method on a finer grid.

Termination criteria

In general, an iterative method should be provided with a good termination criterion to assure that the method stops if the approximate solution is accurate enough. It is important to choose a suitable termination criterion for a given problem which is not too weak or too severe.

An overview of stopping criteria can be found in Vuik [30]. Here we restrict ourselves to the highlights. Sometimes, the iteration method is stopped if the iterations satisfy

$$\|\boldsymbol{y}^{(k)} - \boldsymbol{y}^{(k-1)}\|_2 \leq \varepsilon \tag{3.7}$$

for a fixed tolerance $0 < \varepsilon \ll 1$. Although this criterion seems quite intuitive, one can show, that this is not always a good termination criterion. We will illustrate this for linear convergent iterative methods, *i.e.*, methods for which the iterates satisfy the following equation:

$$\|\boldsymbol{y}^{(k)} - \boldsymbol{y}^{(k-1)}\|_2 \approx r\|\boldsymbol{y}^{(k-1)} - \boldsymbol{y}^{(k-2)}\|_2, \quad r < 1 \ ,$$

and $\boldsymbol{y}^{(k)} \to A^{-1}\boldsymbol{b}$ for $k \to \infty$. An example of a linear convergent method is the SIP method (see Subsection 3.2.2). Now one can show that for a linear convergent method the following equation holds (see for a proof Vuik [30, page 26]):

$$\|\boldsymbol{y} - \boldsymbol{y}^{(k)}\|_2 \leq \frac{r}{1-r}\|\boldsymbol{y}^{(k)} - \boldsymbol{y}^{(k-1)}\|_2 \ .$$

With termination criterion (3.7), it follows that

$$\|\boldsymbol{y} - \boldsymbol{y}^{(k)}\|_2 \leq \frac{r}{1-r}\varepsilon \ ,$$

which tells us that $\|\boldsymbol{y} - \boldsymbol{y}^{(k)}\|_2$ may be large for r close to 1.

For general purposes, it is better to use a termination criterion which is based on the residual, for example

$$\|\boldsymbol{b} - A\boldsymbol{y}^{(k)}\|_2 \leq \varepsilon \ . \tag{3.8}$$

The main disadvantage of this criterion is that it is not scaling invariant. This means that if $\|\boldsymbol{b} - A\boldsymbol{y}^{(k)}\|_2 \leq \varepsilon$, this does not hold for $\|\alpha(\boldsymbol{b} - A\boldsymbol{y}^{(k)})\|_2 \leq \varepsilon$, although the accuracy remains the same. Examples of termination criteria that are scaling invariant are the relative termination criterion, which reads

$$\frac{\|\boldsymbol{b} - A\boldsymbol{y}^{(k)}\|_2}{\|\boldsymbol{b} - A\boldsymbol{y}^{(0)}\|_2} \leq \varepsilon \ , \tag{3.9}$$

and the absolute termination criterion:

$$\frac{\|\boldsymbol{b} - A\boldsymbol{y}^{(k)}\|_2}{\|\boldsymbol{b}\|_2} \leq \varepsilon \ . \tag{3.10}$$

Choosing $\varepsilon \equiv \varepsilon/\kappa_2(A)$ and using the relation $\|AB\|_2 \leq \|A\|_2\|B\|_2$, where $\|\cdot\|_2$ is the matrix norm induced by the 2-norm, it can be easily verified that the relative error satisfies

$$\frac{\|\boldsymbol{y} - \boldsymbol{y}^{(k)}\|_2}{\|\boldsymbol{y}\|_2} \leq \kappa_2(A)\frac{\|\boldsymbol{b} - A\boldsymbol{y}^{(k)}\|_2}{\|\boldsymbol{b}\|_2} \leq \kappa_2(A)\varepsilon \ ,$$

where $\kappa_2(A) = \|A\|_2\|A^{-1}\|_2$ is the condition number of A, which equals λ_n/λ_1 for the case A is symmetric.

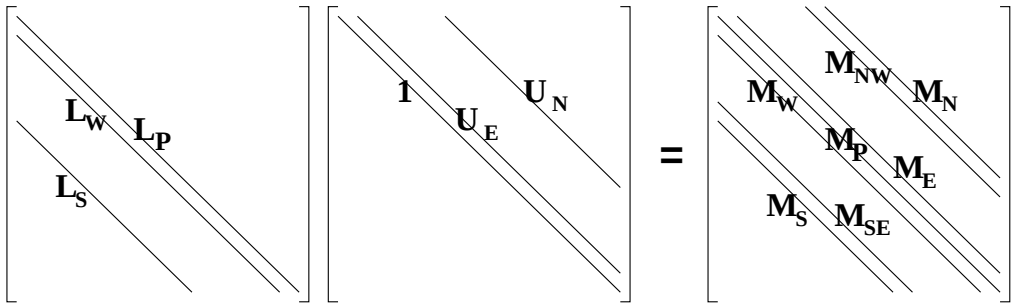

Figure 3.1: Incomplete LU factorization for the SIP method.

3.2.2 SIP basic iterative method

The Strongly Implicit Procedure (SIP) is described in, for example, Ferziger & Perić [4] and Kim & Lee [10]. This method is especially designed for algebraic equations that are discretizations of PDEs and cannot be applied to generic systems of equations.

The SIP method is a BIM (3.2) which is based on the splitting $A = M - N$, with M being an incomplete LU (ILU) factorization of A. This means that a lower triangular matrix L_I and an upper triangular matrix U_I with $\mathrm{diag}(U_\mathrm{I}) = I$ are constructed, such that

$$M = L_\mathrm{I} U_\mathrm{I} \approx A \ ,$$

so the non-zero pattern of $L_\mathrm{I}U_\mathrm{I}$ corresponds to the non-zero pattern of A. Substituting $M = L_\mathrm{I}U_\mathrm{I}$ in the BIM (3.2) yields:

$$\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + U_\mathrm{I}^{-1} L_\mathrm{I}^{-1} (\boldsymbol{b} - A\boldsymbol{y}^{(k-1)}) \ , \tag{3.11}$$

or, equivalently:

for $k = 1, \ldots$ convergence **do**
  $\boldsymbol{r}^{(k-1)} = \boldsymbol{b} - A\boldsymbol{y}^{(k-1)}$;
  Solve $L_\mathrm{I}\boldsymbol{q} = \boldsymbol{r}^{(k-1)}$;
  Solve $U_\mathrm{I}\delta\boldsymbol{y} = \boldsymbol{q}$;
  $\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + \delta\boldsymbol{y}$;
end for

Now the matrices L_I and U_I will be derived for a two-dimensional uniform grid, as defined in Subsection 2.3.2, and a five-point computing molecule. The ILU factorization is of the form as indicated in Figure 3.1, and a row of M corresponding to the (i,j)-th grid point is given by

$$\begin{aligned}
M_\mathrm{S}^{i,j} &= L_\mathrm{S}^{i,j} \ , \\
M_\mathrm{SE}^{i,j} &= L_\mathrm{S}^{i,j} U_\mathrm{E}^{i,j-1} \ , \\
M_\mathrm{W}^{i,j} &= L_\mathrm{W}^{i,j} \ , \\
M_\mathrm{P}^{i,j} &= L_\mathrm{S}^{i,j} U_\mathrm{N}^{i,j-1} + L_\mathrm{W}^{i,j} U_\mathrm{E}^{i-1,j} + L_\mathrm{P}^{i,j} \ , \\
M_\mathrm{E}^{i,j} &= L_\mathrm{P}^{i,j} U_\mathrm{E}^{i,j} \ , \\
M_\mathrm{NW}^{i,j} &= L_\mathrm{W}^{i,j} U_\mathrm{N}^{i-1,j} \ , \\
M_\mathrm{N}^{i,j} &= L_\mathrm{P}^{i,j} U_\mathrm{N}^{i,j} \ .
\end{aligned} \tag{3.12}$$

We now want $N\boldsymbol{y}$ to be small. Writing out the (i,j)-th component of $N\boldsymbol{y}$ results in

$$\begin{aligned}(N\boldsymbol{y})_{i,j} = N_{\mathrm{P}}^{i,j} y_{i,j} + N_{\mathrm{S}}^{i,j} y_{i,j-1} + N_{\mathrm{W}}^{i,j} y_{i-1,j} + N_{\mathrm{E}}^{i,j} y_{i+1,j} + N_{\mathrm{N}}^{i,j} y_{i,j+1} \\ M_{\mathrm{SE}}^{i,j} y_{i+1,j-1} + M_{\mathrm{NW}}^{i,j} y_{i-1,j+1} \ .\end{aligned} \tag{3.13}$$

Now $y_{i+1,j-1}$ and $y_{i-1,j+1}$ will be approximated using Taylor series expansion. For $y_{i+1,j-1}$ this yields

$$\begin{aligned}y_{i+1,j-1} &\approx y_{i,j} + h_1 \frac{dy}{dx_1} - h_2 \frac{dy}{dx_2} \ , \\ &\approx y_{i,j} + h_1 \frac{y_{i+1,j} - y_{i,j}}{h_1} - h_2 \frac{y_{i,j} - y_{i,j-1}}{h_2} \ , \\ &= y_{i+1,j} + y_{i,j-1} - y_{i,j} \ ,\end{aligned} \tag{3.14}$$

and analogous for $y_{i-1,j+1}$, this gives

$$y_{i-1,j+1} \approx y_{i,j+1} + y_{i-1,j} - y_{i,j} \ . \tag{3.15}$$

Multiplying the RHS of (3.14) and (3.15) with a parameter $0 < \alpha < 1$, results in the following approximations

$$\begin{aligned}y_{i+1,j-1} &\approx \alpha(y_{i+1,j} + y_{i,j-1} - y_{i,j}) \ , \\ y_{i-1,j+1} &\approx \alpha(y_{i,j+1} + y_{i-1,j} - y_{i,j}) \ .\end{aligned}$$

With these approximations, (3.13) can be rewritten as

$$\begin{aligned}(N\boldsymbol{y})_{i,j} \approx (N_{\mathrm{P}}^{i,j} - \alpha M_{\mathrm{SE}}^{i,j} - \alpha M_{\mathrm{NW}}^{i,j}) y_{i,j} + \\ (N_{\mathrm{S}}^{i,j} + \alpha M_{\mathrm{SE}}^{i,j}) y_{i,j-1} + (N_{\mathrm{W}}^{i,j} + \alpha M_{\mathrm{NW}}^{i,j}) y_{i-1,j} + \\ (N_{\mathrm{E}}^{i,j} + \alpha M_{\mathrm{SE}}^{i,j}) y_{i+1,j} + (N_{\mathrm{N}}^{i,j} + \alpha M_{\mathrm{NW}}^{i,j}) y_{i,j+1} \ .\end{aligned} \tag{3.16}$$

Setting each of the coefficients in the RHS of (3.16) to zero, then it follows from (3.12) that the entries of N can be expressed in those of L_{I} and U_{I}:

$$\begin{aligned}N_{\mathrm{S}}^{i,j} &= -\alpha M_{\mathrm{SE}}^{i,j} = -\alpha L_{\mathrm{S}}^{i,j} U_{\mathrm{E}}^{i,j-1} \ , \\ N_{\mathrm{W}}^{i,j} &= -\alpha M_{\mathrm{NW}}^{i,j} = -\alpha L_{\mathrm{W}}^{i,j} U_{\mathrm{N}}^{i-1,j} \ , \\ N_{\mathrm{P}}^{i,j} &= \alpha(M_{\mathrm{SE}}^{i,j} + M_{\mathrm{NW}}^{i,j}) = \alpha(L_{\mathrm{S}}^{i,j} U_{\mathrm{E}}^{i,j-1} + L_{\mathrm{W}}^{i,j} U_{\mathrm{N}}^{i-1,j}) \ , \\ N_{\mathrm{E}}^{i,j} &= -\alpha M_{\mathrm{SE}}^{i,j} = -\alpha L_{\mathrm{S}}^{i,j} U_{\mathrm{E}}^{i,j-1} \ , \\ N_{\mathrm{N}}^{i,j} &= -\alpha M_{\mathrm{NW}}^{i,j} = -\alpha L_{\mathrm{W}}^{i,j} U_{\mathrm{N}}^{i-1,j} \ .\end{aligned} \tag{3.17}$$

Now, using $M = A + N$, (3.12) and (3.17), this results in

$$\begin{aligned}L_{\mathrm{S}}^{i,j} &= A_{\mathrm{S}}^{i,j} / (1 + \alpha U_{\mathrm{E}}^{i,j-1}) \ , \\ L_{\mathrm{W}}^{i,j} &= A_{\mathrm{W}}^{i,j} / (1 + \alpha U_{\mathrm{N}}^{i-1,j}) \ , \\ L_{\mathrm{P}}^{i,j} &= A_{\mathrm{P}}^{i,j} + \alpha(L_{\mathrm{S}}^{i,j} U_{\mathrm{E}}^{i,j-1} + L_{\mathrm{W}}^{i,j} U_{\mathrm{N}}^{i-1,j}) - L_{\mathrm{S}}^{i,j} U_{\mathrm{N}}^{i,j-1} - L_{\mathrm{W}}^{i,j} U_{\mathrm{N}}^{i-1,j} \ , \\ U_{\mathrm{E}}^{i,j} &= (A_{\mathrm{E}}^{i,j} - \alpha L_{\mathrm{S}}^{i,j} U_{\mathrm{N}}^{i,j-1}) / L_{\mathrm{P}}^{i,j} \ , \\ U_{\mathrm{N}}^{i,j} &= (A_{\mathrm{N}}^{i,j} - \alpha L_{\mathrm{W}}^{i,j} U_{\mathrm{E}}^{i-1,j}) / L_{\mathrm{P}}^{i,j} \ .\end{aligned} \tag{3.18}$$

The approximations in (3.18) are second order accurate when $\alpha = 1$, but for this α the SIP method generally does not converge. Therefore, often $0.92 < \alpha < 0.96$ is chosen. Entries of (3.18) whose indices are outside the index boundaries should be set equal to zero.

In Ferziger & Perić [4, page 120] a nice comparison of the SIP method to other popular iterative solution methods can be found.

3.2.3 Krylov subspace methods

Define the computing work W to solve the linear system $A\boldsymbol{y} = \boldsymbol{b}$, arising from the discretization of an elliptic PDE, as

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

with N the total number of unknowns and α a certain number. Obviously, a method is said to have an optimal efficiency if $\alpha = 1$.

In general, a BIM (3.2) converges slowly ($\alpha \approx 2$), but fortunately they can be accelerated. Two classes of efficient methods to achieve this can be distinguished: multigrid methods and Krylov subspace methods. Multigrid methods bring α down to the ideal case $\alpha = 1$, Krylov subspace methods bring α down close to $\alpha = 1$. Although multigrid methods have an optimal efficiency, they cost considerably more programming effort than Krylov subspace methods. For a survey on (geometric) multigrid methods see Wesseling [36] and Wesseling [35]. Details on Krylov subspace methods can be found in for example Golub & Van Loan [7], Saad [18], van der Vorst [34], Vuik [30] or Vuik [23] (in Dutch).

Basic idea of Krylov methods

Consider Richardson's iteration (3.6). Multiplication by $-\tilde{A}$ and adding $\tilde{\boldsymbol{b}}$ results in

$$\tilde{\boldsymbol{b}} - \tilde{A}\boldsymbol{y}^{(k)} = \tilde{\boldsymbol{b}} - \tilde{A}\boldsymbol{y}^{(k-1)} - A\tilde{\boldsymbol{r}}^{(k-1)} \ ,$$

hence $\tilde{\boldsymbol{r}}^{(k)} = (I - \tilde{A})\tilde{\boldsymbol{r}}^{(k-1)}$. Using this, we observe that (3.6) can be written as

$$\begin{aligned} \boldsymbol{y}^{(k)} &= \boldsymbol{y}^{(0)} + \tilde{\boldsymbol{r}}^{(0)} + \tilde{\boldsymbol{r}}^{(1)} + \cdots + \tilde{\boldsymbol{r}}^{(k-1)} \\ &= \boldsymbol{y}^{(0)} + \textstyle\sum_{i=0}^{k-1} (I - \tilde{A})^i \tilde{\boldsymbol{r}}^{(0)} \\ &\in \boldsymbol{y}^{(0)} + \text{span}\{\tilde{\boldsymbol{r}}^{(0)}, \tilde{A}\tilde{\boldsymbol{r}}^{(0)}, \ldots, \tilde{A}^{k-1}\tilde{\boldsymbol{r}}^{(0)}\} \\ &= \boldsymbol{y}^{(0)} + \mathcal{K}^{(k)}(\tilde{A}; \tilde{\boldsymbol{r}}^{(0)}) \ , \end{aligned} \tag{3.19}$$

where $\mathcal{K}^{(k)}(\tilde{A}; \tilde{\boldsymbol{r}}^{(0)})$ is called the Krylov subspace of dimension k belonging to the matrix $\tilde{A}$ and initial residual $\tilde{\boldsymbol{r}}^{(0)}$. Methods that look for optimal approximations to $\boldsymbol{y} - \boldsymbol{y}^{(0)}$ in $K^{(k)}(\tilde{A}; \tilde{\boldsymbol{r}}^{(0)})$ are called Krylov subspace (projection) methods.

The Generalized Conjugate Residual (GCR) method

Consider the right-preconditioned system

$$AM^{-1}\hat{\boldsymbol{y}} = \boldsymbol{b}, \quad \hat{\boldsymbol{y}} \equiv M\boldsymbol{y} \ . \tag{3.20}$$

The preconditioned Richardson's iteration for this system reads:

$$\hat{\boldsymbol{y}}^{(k)} = \boldsymbol{b} - (I - AM^{-1})\hat{\boldsymbol{y}}^{(k-1)} = \hat{\boldsymbol{y}}^{(k-1)} + \hat{\boldsymbol{r}}^{(k-1)} \ ,$$

with the residual $\hat{\boldsymbol{r}}^{(k)} \equiv \boldsymbol{b} - AM^{-1}\hat{\boldsymbol{y}}^{(k)}$. Multiplication by $-AM^{-1}$ and adding $\boldsymbol{b}$ results in

$$\boldsymbol{b} - AM^{-1}\hat{\boldsymbol{y}}^{(k)} = \boldsymbol{b} - AM^{-1}\hat{\boldsymbol{y}}^{(k-1)} - AM^{-1}\hat{\boldsymbol{r}}^{(k-1)} \ ,$$

hence

$$\hat{\boldsymbol{r}}^{(k)} = (I - AM^{-1})\hat{\boldsymbol{r}}^{(k-1)} = (I - AM^{-1})^k\hat{\boldsymbol{r}}^{(0)} \ .$$

Using this relation, and the fact that $\hat{\boldsymbol{r}}^{(k)} = \boldsymbol{r}^{(k)}$, results in

$$\boldsymbol{r}^{(k)} = \boldsymbol{r}^{(0)} - \boldsymbol{v} \ , \quad \boldsymbol{v} \in V_k \equiv \mathrm{span}\{AM^{-1}\boldsymbol{r}^{(0)}, \ldots, (AM^{-1})^k\boldsymbol{r}^{(0)}\} \ . \tag{3.21}$$

In the GCR method, $\|\boldsymbol{r}^{(k)}\|_2 = \|\boldsymbol{r}^{(0)} - \boldsymbol{v}\|_2$ is minimized over all $\boldsymbol{v} \in V_k$. Since V_k is a linear subspace, we can decompose $\boldsymbol{r}^{(0)}$ uniquely as

$$\boldsymbol{r}^{(0)} = \boldsymbol{r}_v^{(0)} + \boldsymbol{r}_p^{(0)}, \quad \boldsymbol{r}_v^{(0)} \in V_k, \quad \boldsymbol{r}_p^{(0)} \in V_k^{\perp} \ .$$

Writing out

$$\|\boldsymbol{r}^{(0)} - \boldsymbol{v}\|_2^2 = (\boldsymbol{r}_v^{(0)} - \boldsymbol{v} + \boldsymbol{r}_p^{(0)}, \boldsymbol{r}_v^{(0)} - \boldsymbol{v} + \boldsymbol{r}_p^{(0)}) = (\boldsymbol{r}_v^{(0)} - \boldsymbol{v}, \boldsymbol{r}_v^{(0)} - \boldsymbol{v}) + (\boldsymbol{r}_p^{(0)}, \boldsymbol{r}_p^{(0)}) \ ,$$

it can be easily seen that this is minimal when $\boldsymbol{v} = \boldsymbol{r}_v^{(0)}$, and thus $\boldsymbol{r}^{(k)} = \boldsymbol{r}_p^{(0)}$ by (3.21). This means that the following orthogonality relation holds:

$$\boldsymbol{r}^{(k)} \perp V_k \ . \tag{3.22}$$

With this property, a general expression for the residuals $\boldsymbol{r}^{(k)}$ and the iterands $\boldsymbol{y}^{(k)}$ can be determined. Let $\{\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \ldots, \boldsymbol{v}^{(k)}\}$ be an orthonormal basis for V_k, *i.e.*, $(\boldsymbol{v}^{(i)}, \boldsymbol{v}^{(j)}) = \delta_{i,j}$, then

$$\boldsymbol{r}^{(k)} = \boldsymbol{r}^{(0)} - \boldsymbol{r}_v^{(0)} = \boldsymbol{r}^{(0)} - \sum_{j=1}^{k} \gamma_j \boldsymbol{v}^{(j)} \ .$$

With the orthogonality relation (3.22), it follows that $\gamma_j = (\boldsymbol{r}^{(0)}, \boldsymbol{v}^{(k)})$. We can write

$$\gamma_j = (\boldsymbol{r}^{(0)} - \sum_{i=1}^{j-1} (\boldsymbol{r}^{(0)}, \boldsymbol{v}^{(i)})\boldsymbol{v}^{(i)}, \boldsymbol{v}^{(j)}) = (\boldsymbol{r}^{(j-1)}, \boldsymbol{v}^{(j)})$$

and with this

$$\boldsymbol{r}^{(k)} = \boldsymbol{r}^{(k-1)} - (\boldsymbol{r}^{(k-1)}, \boldsymbol{v}^{(k)})\boldsymbol{v}^{(k)} \ .$$

Since $\boldsymbol{y}^{(k)} = A^{-1}(\boldsymbol{b} - \boldsymbol{r}^{(k)})$, it follows that

$$\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + (\boldsymbol{r}^{(k-1)}, \boldsymbol{v}^{(k)})\boldsymbol{s}^{(k)}, \quad \boldsymbol{s}^{(k)} \equiv A^{-1}\boldsymbol{v}^{(k)} \ .$$

The vectors $\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \ldots, \boldsymbol{v}^{(k)}$ and $\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}, \ldots, \boldsymbol{s}^{(k)}$ remain to be determined.

For determining an orthonormal basis, for example classical Gram-Schmidt (CGS) can be used. However, it can be shown that this method is unstable with respect to rounding errors, see Hoffmann [8]. Methods that do not suffer from this problems are, for example,

modified Gram-Schmidt (MGS) and reorthogonalized classical Gram-Schmidt (RCGS), see Vuik & Frank [24] or Vuik *et al.* [29]. During the orthonormalization process, one has to ensure that the condition $A\boldsymbol{s}^{(k)} = \boldsymbol{v}^{(k)}$ is preserved.

It can be easily shown that $\boldsymbol{y}^{(k)}$ is exact when $\boldsymbol{s}^{(k)} = A^{-1}\boldsymbol{r}^{(k-1)}$, and therefore, we determine $\boldsymbol{s}^{(k)}$ by solving $M\boldsymbol{s}^{(k)} = \boldsymbol{r}^{(k-1)}$, where M is a preconditioner of A. All together, the preconditioned GCR method is given by the following algorithm, see Wesseling [36].

Algorithm 3.1 (GCR method). Given a general non-singular matrix A, a vector $\boldsymbol{b}$, a preconditioner M, and an initial guess $\boldsymbol{y}^{(0)}$ ($A\boldsymbol{y}^{(0)} \approx \boldsymbol{b}$), this algorithm solves the linear system $A\boldsymbol{y} = \boldsymbol{b}$.

$\boldsymbol{r}^{(0)} = \boldsymbol{b} - A\boldsymbol{y}^{(0)}$;
for $k = 1, \ldots,$ convergence **do**
  Solve $M\boldsymbol{s}^{(k)} = \boldsymbol{r}^{(k-1)}$;
  $\boldsymbol{v}^{(k)} = A\boldsymbol{s}^{(k)}$;
  Orthonormalize $\boldsymbol{s}^{(k)}$ and $\boldsymbol{v}^{(k)}$ by, for example, MGS (Algorithm 3.2);
  $\beta = (\boldsymbol{r}^{(k-1)}, \boldsymbol{v}^{(k)})$;
  $\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + \beta\boldsymbol{s}^{(k)}$;
  $\boldsymbol{r}^{(k)} = \boldsymbol{r}^{(k-1)} - \beta\boldsymbol{v}^{(k)}$;
end for
$\boldsymbol{y} \approx \boldsymbol{y}^{(k)}$;

The MGS method, which could be used in the GCR method, is given by the following algorithm.

Algorithm 3.2 (MGS). Given the vectors $\boldsymbol{s}^{(k)}$ and $\boldsymbol{v}^{(k)}$, the sets $\{\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(k-1)}\}$ and $\{\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(k-1)}\}$, this algorithm computes an orthonormal basis for $\{\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(k)}\}$, preserving the relation $A\boldsymbol{s}^{(k)} = \boldsymbol{v}^{(k)}$.

for $j = 1, \ldots, k-1$ **do**
  $\alpha = (\boldsymbol{v}^{(k)}, \boldsymbol{v}^{(j)})$;
  $\boldsymbol{v}^{(k)} := \boldsymbol{v}^{(k)} - \alpha\boldsymbol{v}^{(j)}$;
  $\boldsymbol{s}^{(k)} := \boldsymbol{s}^{(k)} - \alpha\boldsymbol{s}^{(j)}$;
end for
$\boldsymbol{v}^{(k)} := \boldsymbol{v}^{(k)}/\|\boldsymbol{v}^{(k)}\|_2$;
$\boldsymbol{s}^{(k)} := \boldsymbol{s}^{(k)}/\|\boldsymbol{s}^{(k)}\|_2$;

The RCGS orthonormalization method is given by Algorithm B.1 in Appendix B.

Computational work

The orthonormalization process used in Algorithm 3.1 could make the GCR algorithm expensive. First of all, the vectors $\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(k)}$ and $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(k)}$ 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 computational work can be reduced by applying the following techniques:

- Restarting: restart the GCR algorithm after k_{re} iterations by removing $\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(k_{\mathrm{re}})}$ and $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(k_{\mathrm{re}})}$.

- Truncation: allow only k_{tr} vectors $\boldsymbol{v}^{(j)}$ and $\boldsymbol{s}^{(j)}$, and replace the oldest vector by the newest.

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

Robustness and convergence

Inspection of the GCR algorithm in combination with MGS orthonormalization shows that break-down can occur if $\|\boldsymbol{v}^{(k)}\|_2 = 0$ or $\|\boldsymbol{s}^{(k)}\|_2 = 0$. This can happen if $\boldsymbol{r}^{(k-1)} = \boldsymbol{r}^{(k-2)}$, which is unlikely to happen in practice, unless the exact solution is reached, *i.e.*, $\boldsymbol{y}^{(k-1)} = \boldsymbol{y}$. This means that the GCR method is very robust.

The preconditioned Conjugate Gradient (CG) method

In the special case that A is a symmetric positive definite (SPD) matrix, using the so-called preconditioned conjugate gradient (CG) method is less expensive than the GCR method. Compared to the GCR method, many references on the CG method can be found, see for example Golub & Van Loan [7], Saad [18] van der Vorst [34] or Wesseling [36]. The CG method is given by the following algorithm.

Algorithm 3.3 (CG method). Given a SPD $n \times n$ matrix A, a vector $\boldsymbol{b}$, a SPD preconditioner M, and an initial guess $\boldsymbol{y}^{(0)}$ ($A\boldsymbol{y}^{(0)} \approx \boldsymbol{b}$), this algorithm solves the linear system $A\boldsymbol{y} = \boldsymbol{b}$.

$\boldsymbol{r}^{(0)} = \boldsymbol{b} - A\boldsymbol{y}^{(0)}$;
Solve $M\boldsymbol{q}_1 = \boldsymbol{r}^{(0)}$;
$\boldsymbol{s}^{(0)} = \boldsymbol{q}_1$;
for $k = 1, \ldots,$ convergence **do**
  $\alpha = (\boldsymbol{r}^{(k-1)}, \boldsymbol{q}_1)/(A\boldsymbol{s}^{(k-1)}, \boldsymbol{s}^{(k-1)})$;
  $\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + \alpha\boldsymbol{s}^{(k-1)}$;
  $\boldsymbol{r}^{(k)} = \boldsymbol{r}^{(k-1)} - \alpha A\boldsymbol{s}^{(k-1)}$;
  Solve $M\boldsymbol{q}_2 = \boldsymbol{r}^{(k)}$;
  $\beta = (\boldsymbol{r}^{(k)}, \boldsymbol{q}_2)/(\boldsymbol{r}^{(k-1)}, \boldsymbol{q}_1)$;
  $\boldsymbol{s}^{(k)} = \boldsymbol{q}_2 + \beta\boldsymbol{s}^{(k-1)}$;
  $\boldsymbol{q}_1 := \boldsymbol{q}_2$;
end for
$\boldsymbol{y} \approx \boldsymbol{y}^{(k)}$;

Computational work, robustness and convergence

Since no orthonormalization is needed, an iteration with the CG method is less expensive compared to the GCR method, and less memory is required.

Break-down does not occur in computing α and β, because A and M are SPD. Therefore the CG solution method is robust.

Concerning convergence, one can show that after k iterations, the error is bounded by (see Saad [18, page 192] for a proof):

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

where $\kappa = \kappa_2(M^{-1}A) = \lambda_n/\lambda_1$ is the spectral condition number of $M^{-1}A$ and the norm $\|\boldsymbol{y}\|_{M^{-1}A}$ is given by $\|\boldsymbol{y}\|_{M^{-1}A} \equiv (M^{-1}A\boldsymbol{y}, \boldsymbol{y})^{\frac{1}{2}}$.

Choice of Krylov subspace method

For SPD matrices, the CG Krylov subspace method is the most efficient choice. For general matrices, it is more difficult to decide which Krylov subspace method to use, because one can choose from various methods all having their own advantages and disadvantages. In general, the choice of a method will depend on the type of problem to solve and on the computer architecture.

The GCR method described above is very robust, but the orthonormalization process could make the GCR method expensive. A method, which produces the same iterates as GCR and is less expensive, is the generalized minimal residual (GMRES) method, see for example Golub & Van Loan [7], Saad [18], or Wesseling [36]. However, the GCR method has two main advantages compared to GMRES:

1. The GCR method allows a variable preconditioner, GMRES does not.
2. The GCR method can be easily truncated, GMRES can not be easily truncated.

The first property is very important, because in practice systems involving the preconditioner M are usually solved approximately, and therefore the preconditioning can be different in every iteration, see for example Brakkee [2] or Vuik & Saghir [31].

The second property is of importance, because in general truncated GCR converges better than restarted GMRES, especially when superlinear convergence occurs. See Vuik [30] for more details.

3.2.4 Deflation method

Basic idea of deflation

In many problems, the convergence of iterative methods can be significantly slowed down by the presence of several very small eigenvalues in the spectrum of the algebraic system of equations to be solved. An example is given in Vuik *et al.* [32] and Vuik *et al.* [33], where a porous media flow is simulated and a diffusion equation with extreme contrasts in the problem coefficients is aimed to be solved. Discretization results in very ill-conditioned matrices for which the CG method gives erroneous results.

Two preconditioning methods can be distinguished to correct the low-frequency eigenmode components in the spectrum: coarse grid correction and deflation. Coarse grid correction is the oldest of the two methods and has its origin from multigrid solution methods, see Wesseling [36]. A more recently developed method is the so-called deflation technique, which is

originally proposed in Nicolaides [14]. Although both methods are quite similar, it can be proven that, applying a deflation preconditioner results in a matrix with the most favorable eigenvalue spectrum, see Nabben & Vuik [13]. From now on we restrict ourselves entirely to the deflation method.

Consider the general system

$$A\boldsymbol{y} = \boldsymbol{b} \ , \tag{3.23}$$

and let the matrix Z be $n \times m$, where $m \leq n$. Furthermore, let Z be of rank m, *i.e.* Z has linear independent columns. The columns of Z, the so-called deflation vectors, span the deflation subspace, *i.e.*, the space that approximates the eigenspace belonging to the smallest eigenvalues and which is to be projected out of the residual. To do this we define the projectors

$$\begin{aligned} P &= I - AZE^{-1}Z^T \ , \\ Q &= I - ZE^{-1}Z^TA \ , \end{aligned} \tag{3.24}$$

where the $m \times m$ matrix E is defined as $E = Z^TAZ$, and I is the identity matrix of appropriate size. To solve the system (3.23) using deflation, we write $\boldsymbol{y} = (I - Q)\boldsymbol{y} + Q\boldsymbol{y}$, and since

$$(I - Q)\boldsymbol{y} = Z(Z^TAZ)^{-1}Z^TA\boldsymbol{y} = ZE^{-1}Z^T\boldsymbol{b} \ , \tag{3.25}$$

can be computed immediately, we only need to compute $Q\boldsymbol{y}$. In light of the identity $AQ = PA$ we can solve the system

$$PA\tilde{\boldsymbol{y}} = P\boldsymbol{b} \tag{3.26}$$

for $\tilde{\boldsymbol{y}}$, premultiply this by Q and add it to (3.25). We will refer to this system as the deflation system.

Since $PAZ = 0$, this system is obvious singular, and therefore no unique solution exists. However, it can be shown that $Q\boldsymbol{y}$ is unique (see for a proof Vermolen & Vuik [21] for the case that A is SPD).

In Kaasschieter [9] and van der Vorst [34], it is noted that a positive semidefinite system can be solved as long as it is consistent, *i.e.*, as long as its RHS does not has components in the null space span$\{Z\}$ of PA. This is the case for (3.26), because the same projection is applied to both sides of the non-singular system. In van der Vorst [34, page 147], it is shown that a Krylov subspace method could be used to solve (3.26), because the null-space never enters the iteration, and therefore the corresponding zero-eigenvalues do not influence the convergence. Motivated by this fact, we define the effective condition number of the positive semidefinite matrix PA by the ratio of its largest to smallest positive eigenvalue:

$$\kappa_{\text{eff}}(PA) = \frac{\lambda_n}{\lambda_{m+1}} \ .$$

Spectrum of the deflation matrix

Let $\lambda_1 \leq \cdots \leq \lambda_n$ be the eigenvalues of A. Choose the eigenvectors $\boldsymbol{v}_j$ of A such that $\boldsymbol{v}_j^T\boldsymbol{v}_i = \delta_{ij}$, and define $Z = [\boldsymbol{v}_1 \cdots \boldsymbol{v}_m]$. Then we will show that the spectrum of PA satisfies

$$\sigma(PA) = \{0, \ldots, 0, \lambda_{m+1}, \ldots, \lambda_n\} \ .$$

This can be seen as follows. For the choice Z containing m eigenvectors, it appears that

$$E = \text{diag}(\lambda_1, \ldots, \lambda_m) \ ,$$

which implies that $P = I - AZE^{-1}Z^T = I - ZZ^T$. Consider $PA\boldsymbol{v}_j = (I - ZZ^T)\lambda_j \boldsymbol{v}_j$ for $j = 1, \ldots, n$. Since $ZZ^T\boldsymbol{v}_j = \boldsymbol{v}_j$, for $j = 1, \ldots, m$ and $ZZ^T\boldsymbol{v}_j = 0$ for $j = m+1, \ldots, n$ it follows that

$$PA\boldsymbol{v}_j = \begin{cases} 0, & \text{for } j = 1, \ldots, m \ , \\ \lambda_j \boldsymbol{v}_j, & \text{for } j = m+1, \ldots, n \ . \end{cases}$$

This shows that eigenvector deflation cancels the smallest m eigenvalues of the spectrum of A, leaving the rest of the spectrum untouched.

Deflation and preconditioning

Deflation can also be combined with preconditioning, see Subsection 3.2.1. Suppose M is a suitable preconditioner of A, then (3.26) can be replaced by: solve $\tilde{\boldsymbol{y}}$ from the left-preconditioned system

$$M^{-1}PA\tilde{\boldsymbol{y}} = M^{-1}P\boldsymbol{b}$$

and form $Q\tilde{\boldsymbol{y}}$, or solve $\tilde{\boldsymbol{y}}$ from the right-preconditioned system

$$PAM^{-1}\tilde{\boldsymbol{y}} = P\boldsymbol{b} \ ,$$

and form $QM^{-1}\tilde{\boldsymbol{y}}$. Both systems can be solved by a Krylov subspace method, for example by the GCR method.

Starting vectors

When the deflation system (3.26) is solved with an iterative solution method, a starting vector is needed, see Subsection 3.2.1. A suitable choice is to take $\tilde{\boldsymbol{y}}^{(0)} = 0$, because no further information is given.

When an inital guess $\boldsymbol{y}^{(0)}$ is given for the system (3.23), we solve

$$A\boldsymbol{u} = \boldsymbol{f} \ ,$$

where $\boldsymbol{u} \equiv \boldsymbol{y} - \boldsymbol{y}^{(0)}$ and $\boldsymbol{f} \equiv \boldsymbol{b} - A\boldsymbol{y}^{(0)}$, and therefore we solve the deflation system

$$PA\tilde{\boldsymbol{u}} = P\boldsymbol{f} \ ,$$

for $\tilde{\boldsymbol{u}}$ and finally construct

$$\boldsymbol{y} = (I - Q)\boldsymbol{u} + Q\tilde{\boldsymbol{u}} + \boldsymbol{y}^{(0)} \ .$$

Solving systems involving the matrix E

When the system (3.26) is solved with a BIM or a Krylov subspace method, this involves the computation of E^{-1}, for example statements like

$$\boldsymbol{f} = E^{-1}\boldsymbol{g} \tag{3.27}$$

have to be evaluated. In Nabben & Vuik [13, Section 3], it is shown that deflation works correctly as long as E^{-1} is computed with a high accuracy. Since we never determine E^{-1} explicitly, this means that (3.27) is computed by solving $E\boldsymbol{f} = \boldsymbol{g}$ with a direct method, for example by a simple LU factorization (see for example Golub & Van Loan [7, page 152]):

1. Compute $E = LU$;
2. Solve $L\boldsymbol{q} = \boldsymbol{g}$;
3. Solve $U\boldsymbol{f} = \boldsymbol{q}$;

For completeness, the algorithms for evaluating these steps are given in Appendix B, by respectively Algorithms B.2, B.3 and B.4.

Deflated iterative solution methods

The deflation method applied to the BIM (3.2) results into the following algorithm.

Algorithm 3.4 (Deflated BIM). Given a general matrix A, a vector $\boldsymbol{b}$, a preconditioner M, the projectors P and Q as in (3.24), and an initial guess $\boldsymbol{y}^{(0)}$ ($A\boldsymbol{y}^{(0)} \approx \boldsymbol{b}$). This algorithm solves the linear system $A\boldsymbol{y} = \boldsymbol{b}$.

$\boldsymbol{r}^{(0)} = \boldsymbol{b} - A\boldsymbol{y}^{(0)}$
$\tilde{\boldsymbol{y}}^{(0)} = \boldsymbol{0}$;
$\tilde{\boldsymbol{r}}^{(0)} = P\boldsymbol{r}^{(0)}$
for $k = 1, \ldots,$ convergence **do**
  $\delta\tilde{\boldsymbol{y}} = M^{-1}\tilde{\boldsymbol{r}}^{(k-1)}$;
  $\tilde{\boldsymbol{y}}^{(k)} = \tilde{\boldsymbol{y}}^{(k-1)} + \delta\tilde{\boldsymbol{y}}$;
  $\tilde{\boldsymbol{r}}^{(k)} = P\boldsymbol{b} - PA\tilde{\boldsymbol{y}}^{(k)}$;
end for
$\boldsymbol{y} \approx (I - Q)A^{-1}\boldsymbol{r}^{(0)} + Q\tilde{\boldsymbol{y}}^{(k)} + \boldsymbol{y}^{(0)}$;

When deflation is applied to the GCR Krylov subspace method (Algorithm 3.1), this results in the Deflated GCR method (DGCR), given by the following algorithm.

Algorithm 3.5 (Deflated GCR method). Given a general matrix A, a vector $\boldsymbol{b}$, a preconditioner M, the projectors P and Q as in (3.24), and an initial guess $\boldsymbol{y}^{(0)}$ ($A\boldsymbol{y}^{(0)} \approx \boldsymbol{b}$). This algorithm solves the linear system $A\boldsymbol{y} = \boldsymbol{b}$.

$\boldsymbol{r}^{(0)} = \boldsymbol{b} - A\boldsymbol{y}^{(0)}$
$\tilde{\boldsymbol{y}}^{(0)} = \boldsymbol{0}$;
$\tilde{\boldsymbol{r}}^{(0)} = P\boldsymbol{r}^{(0)}$;
for $k = 1, \ldots,$ convergence **do**
  $\boldsymbol{s}^{(k)} = M^{-1}\tilde{\boldsymbol{r}}^{(k-1)}$;
  $\boldsymbol{v}^{(k)} = PA\boldsymbol{s}^{(k)}$
  Orthonormalize $\boldsymbol{s}^{(k)}$ and $\boldsymbol{v}^{(k)}$ by, for example, MGS (Algorithm 3.2);
  $\beta = (\tilde{\boldsymbol{r}}^{(k-1)}, \boldsymbol{v}^{(k)})$;
  $\tilde{\boldsymbol{y}}^{(k)} = \tilde{\boldsymbol{y}}^{(k-1)} + \beta\boldsymbol{s}^{(k)}$;
  $\tilde{\boldsymbol{r}}^{(k)} = \tilde{\boldsymbol{r}}^{(k-1)} - \beta\boldsymbol{v}^{(k)}$;
end for
$\boldsymbol{y} \approx (I - Q)A^{-1}\boldsymbol{r}^{(0)} + Q\tilde{\boldsymbol{y}}^{(k)} + \boldsymbol{y}^{(0)}$;

For the case A is SPD, the CG method (Algorithm 3.3) combined with deflation could be used. Since A is SPD, we can write $Q = P^T$ and therefore the Deflated CG (DCG) algorithm becomes the following.

Algorithm 3.6 (Deflated CG method). Given an $n \times n$ SPD matrix A, a vector $\boldsymbol{b}$, a SPD preconditioner M, the projector P as in (3.24), and an initial guess $\boldsymbol{y}^{(0)}$ ($A\boldsymbol{y}^{(0)} \approx \boldsymbol{b}$). This algorithm solves the linear system $A\boldsymbol{y} = \boldsymbol{b}$.

$\boldsymbol{r}^{(0)} = \boldsymbol{b} - A\boldsymbol{y}^{(0)}$;
$\tilde{\boldsymbol{y}}^{(0)} = \boldsymbol{0}$;
$\tilde{\boldsymbol{r}}^{(0)} = P\boldsymbol{r}^{(0)}$;
$\boldsymbol{p}^{(0)} = \boldsymbol{s}^{(0)} = M^{-1}\tilde{\boldsymbol{r}}^{(0)}$;
for $k = 1, \ldots,$ convergence **do**
  $\alpha = (\tilde{\boldsymbol{r}}^{(k-1)}, \boldsymbol{s}^{(k-1)})/(\boldsymbol{p}^{(k-1)}, PA\boldsymbol{p}^{(k-1)})$;
  $\tilde{\boldsymbol{y}}^{(k)} = \tilde{\boldsymbol{y}}^{(k-1)} + \alpha\boldsymbol{p}^{(k-1)}$;
  $\tilde{\boldsymbol{r}}^{(k)} = \tilde{\boldsymbol{r}}^{(k-1)} - \alpha PA\boldsymbol{p}^{(k-1)}$;
  $\boldsymbol{s}^{(k)} = M^{-1}\tilde{\boldsymbol{r}}^{(k)}$;
  $\beta = (\tilde{r}^{(k)}, \boldsymbol{s}^{(k)})/(\tilde{\boldsymbol{r}}^{(k-1)}, \boldsymbol{s}^{(k-1)})$;
  $\boldsymbol{p}^{(k)} = \boldsymbol{s}^{(k)} + \beta\boldsymbol{p}^{(k-1)}$
end
$\boldsymbol{y} \approx (I - P^T)A^{-1}\boldsymbol{r}^{(0)} + P^T\tilde{\boldsymbol{y}}^{(k)} + \boldsymbol{y}^{(0)}$;

3.3 Solving the stationary incompressible Navier-Stokes equations

There are various methods for iteratively solving the Navier-Stokes equations, see Ferziger & Perić [4] and Wesseling [36]. A popular method used in commercial CFD packages is the Semi-Implicit Method for Pressure-Linked Equations (SIMPLE). Many improved variants of the SIMPLE method have been proposed, for example the more efficient SIMPLE Revised (SIMPLER) method. However, in engineering references, for example Patankar [16] or Ferziger & Perić [4], it is not easy to verify which algebraic systems are actually solved, because the presented algorithms are overrun with details. Therefore, a more mathematical convenient way is to present the methods in a so-called distributive iteration framework.

For simplicity we restrict ourselves to the 2D case ($d = 2$). The DAS (2.25) can be written as the following non-linear system:

$$\begin{bmatrix} N_1 & 0 & G_1 \\ 0 & N_2 & G_2 \\ G_1^T & G_2^T & C \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \boldsymbol{b}_3 \end{bmatrix} \ , \tag{3.28}$$

which we will denote by $\bar{A}\bar{\boldsymbol{y}} = \boldsymbol{b}$. The SIMPLE method can be seen in a distributed iteration framework as follows, see Wesseling [36, Sect. 7.6]. Instead of solving this system, we solve the postconditioned system

$$\bar{A}\bar{B}\hat{\boldsymbol{y}} = \boldsymbol{b} \ , \quad \bar{\boldsymbol{y}} = \bar{B}\hat{\boldsymbol{y}} \ , \tag{3.29}$$

because this system is easier to solve iteratively than $\bar{A}\bar{\boldsymbol{y}} = \boldsymbol{b}$. We choose $\bar{B}$ as

$$\bar{B} = \begin{bmatrix} I & 0 & -N_1^{-1}G_1 \\ 0 & I & -N_2^{-1}G_2 \\ 0 & 0 & I \end{bmatrix} , \tag{3.30}$$

hence, $\bar{A}\bar{B}$ becomes

$$\bar{A}\bar{B} = \begin{bmatrix} N_1 & 0 & 0 \\ 0 & N_2 & 0 \\ G_1^T & G_2^T & C - \sum_{i=1}^{2} G_i^T N_i^{-1} G_i \end{bmatrix} . \tag{3.31}$$

For the iterative solution to (3.29), the splitting

$$\bar{A}\bar{B} = \bar{M} - \bar{T}$$

is introduced, to which the following splitting of the original matrix corresponds:

$$\bar{A} = \bar{M}\bar{B}^{-1} - \bar{T}\bar{B}^{-1} .$$

This leads to the following iterative method for solving $\bar{A}\bar{\boldsymbol{y}} = \boldsymbol{b}$:

$$\bar{\boldsymbol{y}}^{(k)} = \bar{\boldsymbol{y}}^{(k-1)} + \bar{B}\bar{M}^{-1}(\boldsymbol{b} - \bar{A}\bar{\boldsymbol{y}}^{(k-1)}) . \tag{3.32}$$

This iteration is called a distributive iteration, because the correction $\bar{M}^{-1}(\boldsymbol{b} - \bar{A}\bar{\boldsymbol{y}}^{(k-1)})$ is distributed, by multiplication with $\bar{B}$, over the elements of $\boldsymbol{y}^{(k)}$.

In the original SIMPLE method, the matrices $\bar{B}$ and $\bar{M}$ for the iteration (3.32) are chosen as follows. The N_i in (3.31) are obtained by Picard linearization[1], in which the non-linear terms in the momentum equations are linearized by:

$$(u^\alpha u^\beta)_j^{(k)} \approx (u_j^\alpha)^{(k-1)} (u_j^\beta)^{(k)} .$$

As a consequence, $N_i(\boldsymbol{u})$ is replaced by $Q_i^{(k-1)}\boldsymbol{u}_i^{(k)}$, with $Q_i^{(k-1)}$ a matrix that depends on $\boldsymbol{u}^{(k-1)}$. In (3.30) and (3.31), the terms N_i^{-1} are replaced by $N_i^{-1} = D_i^{-1}$, $D_i \equiv \mathrm{diag}(Q_i)$, such that the inverses are easy to compute. The term $C - \sum_{i=1}^{d} G_i^T D_i^{-1} G_i$ in (3.31) is approximated by $-\sum_{i=1}^{d} G_i^T D_i^{-1} G_i$, which we will denote by R for brevity. For these choices, $\bar{B}$ and $\bar{M}$ are approximated by B and M:

$$B = \begin{bmatrix} I & 0 & -D_1^{-1}G_1 \\ 0 & I & -D_2^{-1}G_2 \\ 0 & 0 & I \end{bmatrix} , \qquad M = \begin{bmatrix} Q_1 & 0 & 0 \\ 0 & Q_2 & 0 \\ G_1^T & G_2^T & R \end{bmatrix} , \tag{3.33}$$

and $\bar{A}$ is approximated by A:

$$A = \begin{bmatrix} Q_1 & 0 & G_1 \\ 0 & Q_2 & G_2 \\ G_1^T & G_2^T & C \end{bmatrix} . \tag{3.34}$$

Hence, in each non-linear iteration the postconditoned system

$$AB\hat{\boldsymbol{y}} = \boldsymbol{b} , \quad \boldsymbol{y} = B\hat{\boldsymbol{y}} , \tag{3.35}$$

[1] Another possibility could be, for example, Newton-Raphson linearizaton.

is solved by the iteration process

$$\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + BM^{-1}(\boldsymbol{b} - A\boldsymbol{y}^{(k-1)}) \ , \tag{3.36}$$

with B, M and A given by (3.33) and (3.34). Summarizing, the solution procedure can be written as in the following algorithm.

Algorithm 3.7 (Linearization in the distributed iteration). Let $\bar{\boldsymbol{y}}^{(0)}$ be an inital solution ($\bar{A}\bar{\boldsymbol{y}}^{(0)} \approx \boldsymbol{b}$) to (3.28). Then this algorithm illustrates how linearization is applied in the distributed iteration (3.32).

for $k = 1, \ldots,$ convergence **do**
  $B := B(\boldsymbol{y}^{(k-1)})$;
  $M := M(\boldsymbol{y}^{(k-1)})$;
  Solve the system (3.35); (i)
end
$\bar{\boldsymbol{y}} \approx \boldsymbol{y}$;

This algorithm is equivalent to the original SIMPLE algorithm when we choose B and M according to (3.33) and perform one iteration with (3.36) to evaluate (i). This results in the following algorithm, which can be found in, for example, Patankar [16] or Ferziger & Perić [4].

Algorithm 3.8 (SIMPLE method). Given initial guesses $\boldsymbol{u}^{(0)}$ and $\boldsymbol{p}^{(0)}$ for, respectively, the velocity and pressure field. Then this algorithm solves the non-linear system (3.28) for the stationary incompressible Navier-Stokes equations.

for $k = 1, \ldots,$ convergence **do**
  $Q_i := Q_i(\boldsymbol{u}_i^{(k-1)}), \quad i = 1, \ldots, d;$
  Solve $Q_i \boldsymbol{u}_i^* = \boldsymbol{b}_i - G_i \boldsymbol{p}^{(k-1)}, \quad i = 1, \ldots, d;$ (ii)
  Solve $R\delta\boldsymbol{p} = \boldsymbol{b}_{d+1} - \sum_{i=1}^{d} G_i^T \boldsymbol{u}_i^* - C\boldsymbol{p}^{(k-1)};$ (iii)
  $\boldsymbol{u}_i^{(k)} = \boldsymbol{u}_i^* - D_i^{-1} G_i \delta\boldsymbol{p}, \quad i = 1, \ldots, d;$
  $\boldsymbol{p}^{(k)} = \boldsymbol{p}^{(k-1)} + \delta\boldsymbol{p};$ (iv)
end for

The variables $\boldsymbol{u}_i^*$ and $\delta\boldsymbol{p}$ in this algorithm, we will refer to as the pseudo-velocities and the pressure-correction, respectively.

Underrelaxation

In the SIMPLE method (Algorithm 3.8) corrections of the velocities and the pressure could be large, and therefore underrelaxation is required for the SIMPLE method to converge.

We will now discuss a widely used underrelaxation technique which is discussed in Patankar [16] and Ferziger & Perić [4, page 112]. Rewrite the relaxed BIM (3.3) as follows:

$$M\boldsymbol{y}^{(k)} = (M - \alpha A)\boldsymbol{y}^{(k-1)} + \alpha\boldsymbol{b} \ .$$

Division by α results in

$$\frac{M}{\alpha}\boldsymbol{y}^{(k)} = \frac{M - \alpha A}{\alpha}\boldsymbol{y}^{(k-1)} + \boldsymbol{b} \ . \tag{3.37}$$

Note that A can be decomposed as $A = D - N$, where $D \equiv \mathrm{diag}(A)$ and $-N$ represents the off-diagonal elements. When we choose $M = D - \alpha N$, then Equation (3.37) becomes

$$(\frac{D}{\alpha} - N)\boldsymbol{y}^{(k)} = \frac{1-\alpha}{\alpha} D\boldsymbol{y}^{(k-1)} + \boldsymbol{b} \ . \tag{3.38}$$

Solving this system for $k = 1, 2, \ldots$ generates the iterates. This kind of underrelaxation has the advantage that it makes the matrix more diagonal dominant, which has a positive effect on many iterative solution methods.

Convergence and termination criteria

Since the SIMPLE method can be seen as a distributive (basic iterative) method, SIMPLE will generally converge slowly and the computing work is of $\mathcal{O}(N^2)$. One way to accelerate SIMPLE, which can be integrated in an existing CFD package with relatively low programming effort, is to use a Krylov subspace method. The basic idea is to evaluate (i) in Algorithm 3.7 by performing more than one iteration with a Krylov subspace method.

In Vuik *et al.* [26] and Vuik & Saghir [31], a GCR Krylov subspace acceleration of the SIMPLE(R) method is considered, resulting in the GCR-SIMPLE(R) method. The GCR method can be used, because it allows a variable preconditioner. In Vuik & Saghir [31], results are presented for two industrial furnaces obtained with the TNO code WISH3D-GTM, which has a staggered grid arrangement. For the industrial IFRF furnace, it turns out that GCR-SIMPLE takes approximately a factor 4 less iterations, and costs a factor 2 less computing time, compared to the unaccelerated SIMPLE method.

For colocated grids, a GCR acceleration of the SIMPLE method could also be considered. However, this involves some new research, because no references can be found on how this could be done. A start for this new research is given in Appendix C.

Besides making the residuals small, it is recommendable to make the velocity field also sufficiently divergence free, *i.e.*, to make $\sum_{i=1}^{d} G_i^T \boldsymbol{u}_i^{(k)}$ sufficiently small, for instance

$$\| \sum_{i=1}^{d} G_i^T \boldsymbol{u}_i^{(k)} \|_\infty < \varepsilon V/H \ , \quad 0 < \varepsilon \ll 1 \ ,$$

with $\|\boldsymbol{x}\|_\infty \equiv \max\{|x_1|, \ldots, |x_n|\}$, V and H typical magnitudes for the velocity and domain, and ε a certain fixed tolerance.

3.4 Connection with the X-stream code

The main structure of the X-stream code is given by Figure (3.3). The most outer loop, referred to as time step loop, is only of importance when instationary flow is computed. The middle loop, *i.e.*, the grid level loop, resembles prolongation steps in a multigrid method, see Figure 3.2. The basic idea is to provide a good starting vector for the finest grid by interpolating the solution from a coarser to a finer grid.

In X-stream, the incompressible Navier-Stokes equations are solved with the SIMPLE method, according to Algorithm 3.8. Underrelaxation of the form (3.38) is applied to the systems

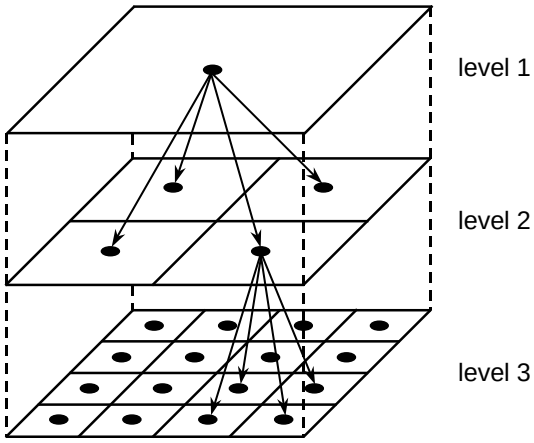

Figure 3.2: Multigrid prolongation operator in X-stream illustrated for three grids varying from coarse (level 1) to fine (level 3).

(*ii*) for solving the pseudo-velocities. For solving the pressure-correction, the update (*iv*) is replaced by

$$\boldsymbol{p}^{(k)} = \boldsymbol{p}^{(k-1)} + \alpha_p \delta \boldsymbol{p} \ ,$$

with $0 < \alpha_p < 1$ a chosen underrelaxation parameter.

The systems in the SIMPLE method can be solved with the SIP method, the CG method and the Space Tri-Diagonal Matrix Algorithm (SPTDMA), see Patankar [16] or Ferziger & Perić [4]. All these iterative solution methods are used within a domain decomposition context, see Chapter 4.

In the X-stream code, a termination criterion of the form (3.8) is used for solving the pressure-correction system. A relative termination criterion (3.9) is used for solving the pseudo-velocities systems, as well as for all the remaining variables. However, a termination criterion for the so-called inner iterations will in general not be satisfied, because usually a predefined number of inner iterations and a very small tolerance is taken.

In X-stream, no explicit termination criterion is evaluated after the velocity and pressure updates. Instead, the SIMPLE algorithm stops when the pseudo-velocities satisfy (3.9) and the RHS of (*iii*) in Algorithm 3.8 is smaller than a given tolerance. The criterion for the pressure is a physical one, because the RHS of (*iii*) represents more or less the divergence of the velocity field, and will decrease to zero when the SIMPLE method converges.

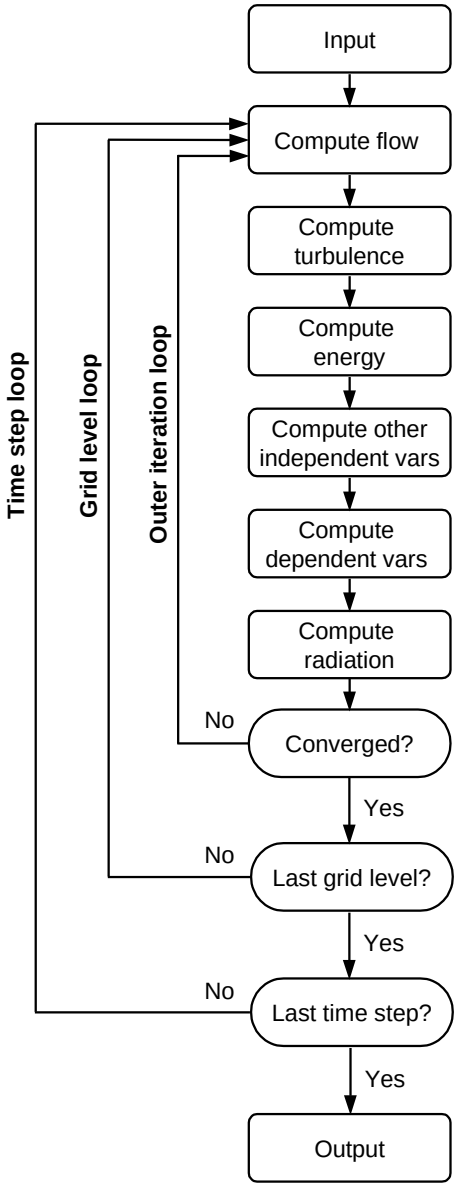

Figure 3.3: Main flowchart of the X-stream code.

4

Domain decomposition methods

4.1 Introduction

The term domain decomposition (DD) is used differently by specialists in numerical analysis of PDEs. In parallel computing, it means distributing data from a computing model over the processors in a distributed memory computer. In asymptotic analysis, DD refers to the determination of which PDEs to solve. In preconditioning methods, DD refers only to the solution method for the algebraic system of equations arising from the discretization. Note that all these three interpretations may actually occur in one problem. We will restrict ourselves to 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 complicated geometry.
- Different physical models can be used in different subdomains.
- Local grid refinement can be easier implemented.
- Reduction of memory requirements, because the subproblem can be much smaller than the total problem.

The structure of this chapter is as follows. In Section 4.2, the Schwarz method is discussed, followed by a discussion of convergence aspects in Section 4.3. In Section 4.4, deflated Krylov-Schwarz methods are treated in a DD context. In Section 4.5, DD for the stationary Navier-Stokes equations is described. Finally, in Section 4.6, the connection with the X-stream code is given.

4.2 Alternating Schwarz methods

A survey on DD methods can be found in, for example, Smith [19], Chan [3], or Saad [18]. Basically two types of DD methods can be distinguished: overlapping and non-overlapping methods. Overlapping DD methods are known as Schwarz alternating methods, which are iterative methods. Non-overlapping methods can be divided into so-called substructuring methods, which are direct methods, and so-called Schur-complement methods, which are iterative methods. Non-overlapping methods differ from overlapping methods by solving an

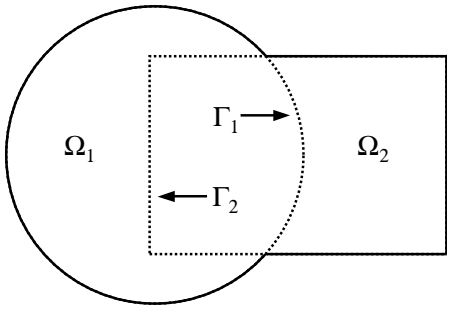

Figure 4.1: An overlapping decomposition of the domain.

additional interface equation. We will restrict ourselves entirely to the Schwarz methods.

The Schwarz method dates from 1870 and was originally intended as an analytical method. In its original form it is known as the multiplicative Schwarz method, which will be described next.

Multiplicative Schwarz method

In general, a DD method aims to solve the differential equation

$$\mathcal{L}y = b \quad \text{in } \Omega \ , \tag{4.1}$$

with suitable BCs on $\partial\Omega$. The domain Ω is decomposed into subdomains $\bar{\Omega} = \bar{\Omega}_1 \cup \cdots \cup \bar{\Omega}_k$, where Ω is open, and Ω_i are open subsets of Ω. The simplest form of $\mathcal{L}$ is minus the Laplacian $-\Delta$, resulting in the Poisson equation. For simplicity, we restrict ourselves to Dirichlet BCs $y = g$ on $\partial\Omega$. Other BCs can be treated with ease. For sake of simplicity we only consider two subdomains $\bar{\Omega} = \bar{\Omega}_1 \cup \bar{\Omega}_2$.

Overlapping Schwarz DD methods use an overlapping decomposition of the domain Ω into subdomains such that $\Omega_1 \cap \Omega_2 \neq \emptyset$. 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, see Figure 4.1.

The alternating Schwarz method begins with selecting an initial guess $y_2^{(0)}$ for the values in Ω_2. Then iteratively for $k = 1, \ldots$ one solves the subproblem,

$$\begin{cases} \mathcal{L}y_1^{(k)} = b & \text{on } \Omega_1 \ , \\ y_1^{(k)} = y_2^{(k-1)} & \text{on } \Gamma_1 \ , \\ y_1^{(k)} = g & \text{on } \partial\Omega_1 \setminus \Gamma_1 \ , \end{cases} \tag{4.2}$$

for $y_1^{(k)}$, followed solving of the subproblem,

$$\begin{cases} \mathcal{L}y_2^{(k)} = b & \text{on } \Omega_2 \ , \\ y_2^{(k)} = y_1^{(k)} & \text{on } \Gamma_2 \ , \\ y_2^{(k)} = g & \text{on } \partial\Omega_2 \setminus \Gamma_2 \ . \end{cases} \tag{4.3}$$

The k-th iterate is then defined by

$$y^{(k)}(x_1, x_2) = \begin{cases} y_1^{(k)}(x_1, x_2) & \text{if } (x_1, x_2) \in \Omega \setminus \Omega_2 \ , \\ y_2^{(k)}(x_1, x_2) & \text{if } (x_1, x_2) \in \Omega_2 \ . \end{cases}$$

It can be shown (see Chan [3]) that for the elliptic operator $\mathcal{L}$, the iterates $\{y^{(k)}\}$ converge linearly to the true solution y on Ω, that is

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

where $\rho < 1$ depends on the choice of Ω_1 and Ω_2.

The discrete form of (4.1) is denoted by

$$A\boldsymbol{y} = \boldsymbol{b} \ , \tag{4.4}$$

where A represents the discretization of the continuous operator $\mathcal{L}$ and the BCs on the total 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 the unknowns in the interior of Ω_1 and Ω_2 respectively. The total number of unknowns is $n = |I| = |I_1 \cup I_2|$ 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 of the matrix R_i^T consist of ones and zeros with at most one '1' in each row. The transpose R_i is a trivial restriction matrix which restricts 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 $R_i^T R_i \neq I_n$. Also note that the matrices R_i are never formed in practice.

Now the local subdomain matrices can be written in terms of the global matrix A and the restriction matrices R_i as

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

The algebraic Schwarz iteration starts with an initial guess $\boldsymbol{y}^{(0)}$ and constructs a sequence of approximations for $k = 1, 2, \ldots$ as follows:

$$\boldsymbol{y}^{(k-\frac{1}{2})} = \boldsymbol{y}^{(k-1)} + R_1^T A_{11}^{-1} R_1 (\boldsymbol{b} - A\boldsymbol{y}^{(k-1)}) \ , \tag{4.5}$$

$$\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-\frac{1}{2})} + R_2^T A_{22}^{-1} R_2 (\boldsymbol{b} - A\boldsymbol{y}^{(k-\frac{1}{2})}) \ . \tag{4.6}$$

For the cell-centered case that the two subdomains have a single grid line in common (see Figure 4.2), $I_1 \cap I_2 = \emptyset$, this iteration is a classical block Gauß-Seidel iteration (see Vuik [30,

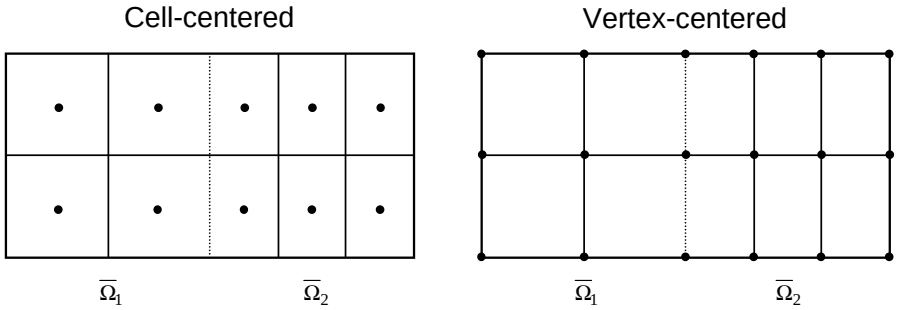

Figure 4.2: Cell-centered and vertex-centered discretization in the case of two subdomains. (• grid points; — finite volume boundaries; - - common grid line.)

page 21]). For $I_1 \cap I_2 \neq \emptyset$, this iteration corresponds to a block Gauß-Seidel iteration with overlapping blocks.

Define the matrix

$$P_i = R_i^T A_{ii}^{-1} R_i A, \quad i = 1, 2 \ ,$$

then the global error $\boldsymbol{e}^{(k)}$ for (4.5)–(4.6) can be written as

$$\boldsymbol{e}^{(k-\frac{1}{2})} = (I - P_1)\boldsymbol{e}^{(k-1)} \ , \tag{4.7}$$

$$\boldsymbol{e}^{(k)} = (I - P_2)\boldsymbol{e}^{(k-\frac{1}{2})} \ . \tag{4.8}$$

The matrices P_i represent projection operators ($P_i^2 = P_i$). Combining (4.7) and (4.8) yields

$$\boldsymbol{e}^{(k)} = (I - P_2)(I - P_1)\boldsymbol{e}^{(k-1)} \ ,$$

thus the iteration matrix is $(I - P_2)(I - P_1)$, explaining why the algorithm is called multiplicative.

The iteration process (4.5)–(4.6) can be written as

$$\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + M_{\text{gs}}^{-1}(\boldsymbol{b} - A\boldsymbol{y}^{(k-1)}) \ , \tag{4.9}$$

with $M_{\text{gs}}^{-1}A \equiv I - (I - P_2)(I - P_1)$. For the case $I_1 \cap I_2 = \emptyset$, iteration (4.9) solves the system

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{bmatrix} \ , \tag{4.10}$$

where A_{11} and A_{22} represent the subdomain discretization matrices and A_{12} and A_{21} represent the coupling between the subdomains. For this case the preconditioner M_{gs} is given by

$$M_{\text{gs}} = \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix} \ .$$

Note that the multiplicative Schwarz method does not lend itself for parallelization, because of the preconditioner.

Additive Schwarz method

The additive Schwarz method is governed by computing the residual for $\boldsymbol{y}^{(k-1)}$ in (4.6),

$$\boldsymbol{y}^{(k-\frac{1}{2})} = \boldsymbol{y}^{(k-1)} + R_1^T A_{11}^{-1} R_1(\boldsymbol{b} - A\boldsymbol{y}^{(k-1)}) \ , \tag{4.11}$$

$$\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-\frac{1}{2})} + R_2^T A_{22}^{-1} R_2(\boldsymbol{b} - A\boldsymbol{y}^{(k-1)}) \ . \tag{4.12}$$

This iteration corresponds to a block Gauß-Jacobi iteration (see Vuik [30]), with overlapping blocks. Writing this in terms of the error $\boldsymbol{e}^{(k)}$ gives

$$\boldsymbol{e}^{(k)} = (I - P_1 - P_2)\boldsymbol{e}^{(k-1)} \ ,$$

which explains why the algorithm is called additive. Rewriting (4.11)–(4.12) results in

$$\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + M_{\text{jac}}^{-1}(\boldsymbol{b} - A\boldsymbol{y}^{(k-1)}) \ , \tag{4.13}$$

with $M_{\text{jac}}^{-1}A \equiv P_1 + P_2$. For the case $I_1 \cap I_2 = \emptyset$, the preconditioner M_{jac} is given by

$$M_{\text{jac}} = \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix} \ . \tag{4.14}$$

The additive Schwarz method lends itself well for parallelization, compared to the multiplicative case.

In general, the Schwarz domain decomposition method can be interpreted as a method for solving the preconditioned system

$$M^{-1}A\boldsymbol{y} = M^{-1}\boldsymbol{b} \ ,$$

with $M^{-1} = M_{\text{jac}}^{-1}$ or $M^{-1} = M_{\text{gs}}^{-1}$ for, respectively, the additive and multiplicative case.

4.3 Convergence aspects

4.3.1 Local coupling

It can be observed that the multiplicative Schwarz method converges approximately twice as fast as the additive Schwarz method, see for example Smith [19] and Verweij [22] for comparisons. However, both Schwarz methods converge slowly, because they can be seen as BIMs, see Subsection 3.2.1.

Therefore, it is advisable to accelerate them, for example by a Krylov subspace method, see Subsection 3.2.3. In general, domain decomposition accelerated by a Krylov subspace method is called a Krylov-Schwarz method. Equivalently, Krylov-Schwarz means that domain decomposition is used as a preconditioner for a Krylov subspace method. Krylov-Schwarz acceleration can be interpreted as a method to increase local coupling, in which each subdomain only interacts through coupling with its neighbors.

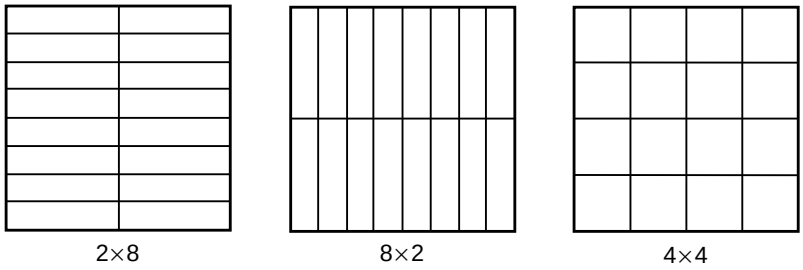

Figure 4.3: Three decompositions of a square into 16 subdomains.

4.3.2 Global coupling

For matrices arising from the discretization of elliptic PDEs, global coupling between the subdomains is important. Elliptic problems have the property that a slight modification in the data at a certain area (for example at a boundary) influences the solution on the entire domain. In terms of the Schwarz method, this means that new information computed at a subdomain has to propagate to all other subdomains in a fast way.

Since in each Schwarz iteration, information can only travel through one interface from a subdomain A to a subdomain B, the convergence rate decreases when the number of subdomains between A and B increases. Several methods exist to increase the global coupling between subdomains.

Choosing the right decomposition

An obvious way to minimize the loss of convergence is to decompose the original domain into a decomposition, which has an equal number of subdomains in every dimension as much as possible. See for example Figure 4.3, in which a square is decomposed into three decompositions. The two decompositions 2×8 and 8×2 have a large number of subdomains in one dimension, and are therefore less favorable than the 4×4 decomposition, which has an equal number of subdomains in every dimension. Another disadvantage of the first two decompositions is that the total interface size is greater than that of the 4×4 decomposition, which results in more communication between processors in a parallel computing environment.

Increasing the size of overlap

One way to increase global coupling is to increase the amount of overlap of the subdomains. However, a drawback of overlapping subdomains is that the computing work increases proportionally to the size of overlap, because in each iteration several unknowns are computed more than once. This is the main reason why many authors consider a simple Schwarz DD with minimal overlap, see Brakkee [2] and Verweij [22]. How this method works, will be illustrated by Figure 4.4.

In this figure a cell-centered discretization is given with boundary nodes, which resembles a colocated grid arrangement, see Figure 2.2. The shaded cells denote ghost cells, also called halo-cells. Let there an initial guess being given on the entire domain. Then the additive Schwarz method goes as follows:

for $k = 1, \ldots,$ convergence **do**
    Build the RHS of the system for subdomain 1 and solve;
    Build the RHS of the system for subdomain 2 and solve;
    Update the ghost cells of subdomain 1 and 2;
end

For the multiplicative case, the solution process is given by:

for $k = 1, \ldots,$ convergence **do**
    Build the RHS of the system for subdomain 1 and solve;
    Update the ghost cells of subdomain 2;
    Build the RHS of the system for subdomain 2 and solve;
    Update the ghost cells of subdomain 1;
end

Hence, the difference between the additive and multiplicative Schwarz method lies in the number of updates in each Schwarz iteration. Since for the additive case the ghost cells are updated after a Schwarz iteration, the subdomain solutions can be computed in parallel on different processors.

Coarse grid correction and deflation

An interpretation for the loss in convergence is, that each subdomain contributes to a small eigenvalue in the spectrum of the global matrix. Therefore, a coarse grid correction preconditioning or a deflation preconditioning can be applied to remove those small eigenvalues, see Subsection 3.2.4. It turns out that coarse grid correction and deflation make the convergence rate more or less independent on the number of subdomains. Because of arguments presented in Nabben & Vuik [13], where deflation is compared to coarse grid correction, we restrict ourselves to the deflation technique.

4.3.3 Inaccurate subdomain solution

In the original Schwarz method, it is assumed that the subdomain problems are solved accurate enough. However, solving the subdomain problems inaccurately can reduce the computing time significantly, see for example Brakkee [2] and Verweij [22].

When solving the subdomain problems inaccurately, one has to be aware that the iteration processes (4.9) and (4.13) are implemented in a correct way. In general, Schwarz domain decomposition is typically implemented as

$$\tilde{M}\boldsymbol{y}^{(k)} = (M - A)\boldsymbol{y}^{(k-1)} + \boldsymbol{b} \ , \tag{4.15}$$

where the first RHS term $(M-A)\boldsymbol{y}^{(k-1)}$ represents the discretization of the internal boundary conditions, which is always exact, and the LHS term $\tilde{M}\boldsymbol{y}^{(k)}$ indicates the inaccurate solution of the subdomain problems using some type of linear system solver. In general, the stationary solution to (4.15) satisfies the disturbed system $(A + \tilde{M} - N)\boldsymbol{y} = \boldsymbol{b}$ instead of $A\boldsymbol{y} = \boldsymbol{b}$. Therefore, with inaccurate subdomain solutions, the difference between $\tilde{M}$ and M may be quite large, and the computed solution $\boldsymbol{y}$ may have a large error. Instead of (4.15), we must use the iteration

$$\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + \tilde{M}^{-1}(\boldsymbol{b} - A\boldsymbol{y}^{(k-1)}) \ .$$

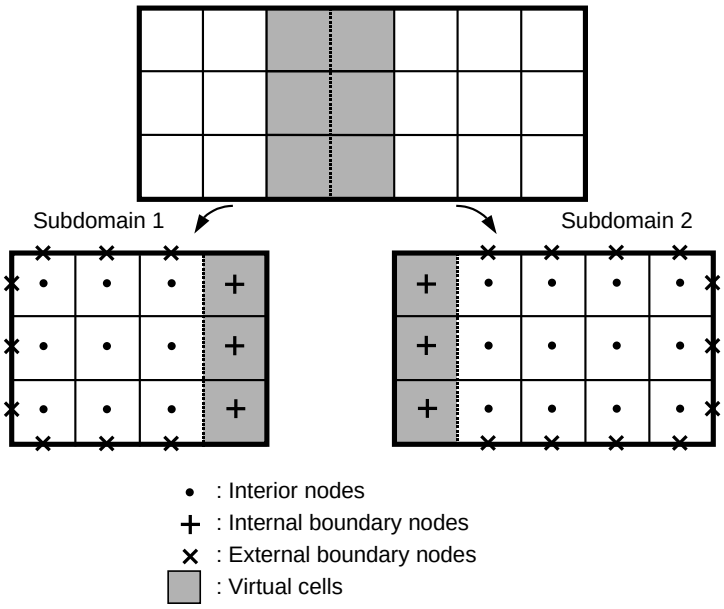

Figure 4.4: Decomposition of the domain into two subdomains. The dashed line denotes the block interface and the shaded cells are the ghost cells.

With inaccurate subdomain solution, we have

$$\tilde{M} = \tilde{M}_{\text{gs}} = \begin{bmatrix} \tilde{A}_{11} & 0 \\ A_{21} & \tilde{A}_{22} \end{bmatrix} \quad \text{or} \quad \tilde{M} = \tilde{M}_{\text{jac}} = \begin{bmatrix} \tilde{A}_{11} & 0 \\ 0 & \tilde{A}_{22} \end{bmatrix} .$$

When we use a Krylov-Schwarz method with inaccurate subdomain solution, two main strategies can be distinguished for approximating the subdomain solutions:

1. Apply one or a few iterations of a linear convergent BIM, for instance, of the SIP method.

2. Solve the subdomain problems with a (preconditioned) Krylov subspace method to a fixed tolerance, to a variable tolerance, or for a predetermined number of iterations.

It can be shown that strategy 2 can only be applied to a Krylov-Schwarz method, which allows a variable preconditioner. For example, the GCR-Schwarz method is such a method, while the GMRES-Schwarz method is not, see Brakkee [2] for more details.

4.4 Deflated Krylov-Schwarz domain decomposition

Motivated by the previous section, we restrict ourselves to a simple cell-centered additive Schwarz DD method with minimal overlap. Furthermore, we apply the following methods for improving the convergence rate and reducing the computing time of the Schwarz method:

- Accelerate the Schwarz method with minimal overlap by a Krylov subspace method.

- Apply the deflation technique to make the convergence rate of the DD independent on the number of subdomains.

- Solve the subdomain problems inaccurately with a predefined number of iterations of a basic iterative method.

For general matrices, a suitable Krylov subspace method is the GCR method, see Subsection 3.2.4. Combined with deflation and with inaccurate subdomain solution, the Deflated GCR method (Algorithm 3.5) can be rewritten in a domain decomposition context as follows.

Algorithm 4.1 ((Deflated) GCR-Schwarz method). Given a general matrix A, and the vectors $\boldsymbol{y}$ and $\boldsymbol{b}$, such that the system $A\boldsymbol{y} = \boldsymbol{b}$ is of the form (4.10). Let $\boldsymbol{y}^{(0)}$ be an initial guess such that $A\boldsymbol{y}^{(0)} \approx \boldsymbol{b}$. For deflation, define the matrix Z. Then this algorithm accelerates the additive Schwarz method with minimal overlap for solving the linear system $A\boldsymbol{y} = \boldsymbol{b}$.

$\boldsymbol{r}^{(0)} = \boldsymbol{b} - A\boldsymbol{y}^{(0)}$;
Solve $Z^T AZ\boldsymbol{q}_1 = Z^T \boldsymbol{r}^{(0)}$; // Contribution deflation
$\boldsymbol{r}^{(0)} := \boldsymbol{r}^{(0)} - AZ\boldsymbol{q}_1$; // Contribution deflation
$\boldsymbol{y}^{(0)} = \boldsymbol{0}$; // Contribution deflation
for $k = 1, \ldots,$ ngcrschw **do**
  Solve $A_{mm}\boldsymbol{s}_m^{(k)} = \boldsymbol{r}^{(k-1)}$ inaccurately, $m = 1, \ldots, nblock$;
  $\boldsymbol{v}^{(k)} = A\boldsymbol{s}^{(k)}$;
  Solve $Z^T AZ\boldsymbol{q}_2 = Z^T \boldsymbol{v}^{(k)}$; // Contribution deflation
  $\boldsymbol{v}^{(k)} := \boldsymbol{v}^{(k)} - AZ\boldsymbol{q}_2$; // Contribution deflation
  Orthonormalize $\boldsymbol{s}^{(k)}$ and $\boldsymbol{v}^{(k)}$ by applying, for example, MGS (Algorithm 3.2);
  $\beta = (\boldsymbol{r}^{(k-1)}, \boldsymbol{v}^{(k)})$;
  $\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + \beta\boldsymbol{s}^{(k)}$;
  $\boldsymbol{r}^{(k)} = \boldsymbol{r}^{(k-1)} - \beta\boldsymbol{v}^{(k)}$;
end for
$\boldsymbol{y} \approx \boldsymbol{y}^{(k)}$
Solve $Z^T AZ\boldsymbol{q}_2 = Z^T A\boldsymbol{y}^{(k-1)}$; // Contribution deflation
$\boldsymbol{y} \approx \boldsymbol{y}^{(k-1)} + \boldsymbol{y}^{(0)} + Z(\boldsymbol{q}_1 - \boldsymbol{q}_2)$; // Contribution deflation

The GCR-Schwarz algorithm without deflation is obtained by removing the statements provided with the comment 'Contribution deflation'.

For the case that the matrix is SPD, the CG Krylov subspace method is most suitable, see Subsection 3.2.3. Rewriting Algorithm 3.6, results into the following algorithm.

Algorithm 4.2 ((Deflated) CG-Schwarz method). Given a SPD matrix A, the vectors $\boldsymbol{y}$ and $\boldsymbol{b}$, such that the system $A\boldsymbol{y} = \boldsymbol{b}$ is of the form (4.10). Let $\boldsymbol{y}^{(0)}$ be an initial guess such that $A\boldsymbol{y}^{(0)} \approx \boldsymbol{b}$. For deflation define the matrix Z. Then this algorithm accelerates the additive Schwarz method with minimal overlap for solving the linear system $A\boldsymbol{y} = \boldsymbol{b}$.

$\boldsymbol{r}^{(0)} = \boldsymbol{b} - A\boldsymbol{y}^{(0)}$;
Solve $Z^T AZ\boldsymbol{q}_1 = Z^T \boldsymbol{r}^{(0)}$; // Contribution deflation
$\boldsymbol{r}^{(0)} := \boldsymbol{r}^{(0)} - AZ\boldsymbol{q}_1$; // Contribution deflation
$\boldsymbol{y}^{(0)} = \boldsymbol{0}$; // Contribution deflation
Solve $A_{mm}\boldsymbol{s}_m^{(0)} = \boldsymbol{r}^{(0)}$ inaccurately, $m = 1, \ldots, nblock$;
$\boldsymbol{p}^{(0)} = \boldsymbol{s}^{(0)}$;
for $k = 1, \ldots,$ ncgschw **do**
  $\boldsymbol{v}^{(k-1)} = A\boldsymbol{p}^{(k-1)}$;

Solve $Z^T A Z \boldsymbol{q}_2 = Z^T \boldsymbol{v}^{(k-1)}$; // Contribution deflation
$\boldsymbol{v}^{(k-1)} := \boldsymbol{v}^{(k-1)} - AZ\boldsymbol{q}_2$; // Contribution deflation
$\alpha = (\boldsymbol{r}^{(k-1)}, \boldsymbol{s}^{(k-1)})/(\boldsymbol{p}^{(k-1)}, \boldsymbol{v}^{(k-1)})$
$\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + \alpha \boldsymbol{p}^{(k-1)}$;
$\boldsymbol{r}^{(k)} = \boldsymbol{r}^{(k-1)} - \alpha \boldsymbol{v}^{(k-1)}$;
Solve $A_{mm} \boldsymbol{s}_m^{(k)} = \boldsymbol{r}^{(k)}$ inaccurately, $m = 1, \ldots, nblock$;
$\beta = (\boldsymbol{r}^{(k)}, \boldsymbol{s}^{(k)})/(\boldsymbol{r}^{(k-1)}, \boldsymbol{s}^{(k-1)})$;
$\boldsymbol{p}^{(k)} = \boldsymbol{s}^{(k)} + \beta \boldsymbol{p}^{(k-1)}$;
end for
$\boldsymbol{y} \approx \boldsymbol{y}^{(k)}$
Solve $Z^T A Z \boldsymbol{q}_2 = Z^T A \boldsymbol{y}^{(k)}$; // Contribution deflation
$\boldsymbol{y} \approx \boldsymbol{y}^{(k)} + \boldsymbol{y}^{(0)} + Z(\boldsymbol{q}_1 - \boldsymbol{q}_2)$; // Contribution deflation

Note that till now, we have said nothing about the choice of the matrix Z. Several authors consider so-called subdomain deflation, see for example Nicolaides [14], Mansfield [12] and Vuik & Frank [28]. For this choice, each coefficient of the matrix Z is given by

$$z_{ij} = \begin{cases} 1 , & i \in I_j , \\ 0 , & i \notin I_j , \end{cases} \tag{4.16}$$

where I_j is the index set of the unknowns in the interior of subdomain Ω_j. For example, consider the case of two subdomains, each containing n grid cells. Then Z is $2n \times 2$ and given by

$$Z = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} ,$$

where $\mathbf{1}$ and $\mathbf{0}$ denote vectors of length n, containing respectively all ones and all zeros.

Although in literature, this kind of deflation is usually referred to as subdomain deflation, we will call this constant deflation (CD) from now on. For constant deflation, we will refer to the DGCR-Schwarz method (Algorithm 4.1) and DCG-Schwarz method (Algorithm 4.2) as respectively CDGCR-Schwarz and CDCG-Schwarz, or simply as CDGCR and CDCG.

In Chapter 5 we will go into more detail on choosing the deflation vectors.

A note on Krylov subspace acceleration and parallel computing

In parallel computing several aspects are of importance, for instance: speedup, efficiency, scalability, communication, synchronization, grain size, load balance, etc. We will not go in details on parallel computing, but refer to Kumar [11] or Vuik [30] for an introduction.

Since Krylov subspace methods mainly consist of matrix-vector multiplications and inner products, this results in additional communication between processors compared to the unaccelerated Schwarz method.

Matrix-vector multiplications can be done in parallel by local (nearest-neighbor) communication, in which each processor communicates with its neighbor. For inner products, each processor has to communicate with all processors, which makes global communication most expensive. Therefore, inner products should be avoided as much as possible.

For the GCR Krylov subspace method, the number of inner products can be reduced by truncation or restarting, see Subsection 3.2.3. When the interconnection network is slow, one could also choose for the RCGS orthonormalization method (Algorithm B.1), see Frank & Vuik [5], Vuik & Frank [24] or Vuik *et al.* [29].

4.5 Domain decomposition for the incompressible Navier-Stokes equations

Domain decomposition for the SIMPLE method, given by Algorithm 3.8, can be achieved in a straightforward way. In every iteration of the SIMPLE method, in total $d + 1$ systems of equations have to be solved, where d is the spatial dimension: d for solving the pseudo-velocities and 1 for solving the pressure-correction.

For the Schwarz method with minimal overlap, the systems in Algorithm 3.8 are of the form (4.10) with $A = Q_i$ from (ii) and $\boldsymbol{y} = \boldsymbol{u}_i^*$, or $A = R$ from (iii) and $\boldsymbol{y} = \delta\boldsymbol{p}$.

All the systems involved can be solved with the Schwarz method. Since in general, the matrix Q_i is non-symmetric we could also use the GCR-Schwarz Krylov subspace method (Algorithm 4.1) to solve the pseudo -velocities. However, generally the systems involving Q_i are not very difficult to solve.

The system for solving the pressure-correction is generally most time-consuming in the SIMPLE method, because the corresponding matrix R resembles a discrete Poisson operator. By arguments discussed in Section 4.3, we can use a Deflated Krylov-Schwarz subspace method to solve this system. If R is symmetric, the DCG-Schwarz method (Algorithm 4.2) is likely to be efficient. If R is non-symmetric, the DGCR-Schwarz method (Algorithm 4.1) could be chosen.

4.6 Connection with the X-stream code

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

- Unaccelerated additive Schwarz method with minimal overlap.
- Inaccurate subdomain solution according to iteration (4.15) by a single iteration of SIP, SPTDMA or CG.
- Block-structured grid.
- Local grid refinement can be done at block level.
- Application of different models on different blocks.
- Parallellized using the MPI (Message Passing Interface) standard.

For more details and an implementation description, see Verweij [22].

In X-stream, a set of subdomains (blocks) for which a 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 4.5. 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. Although this method appears quite effective, no references could be found on this method.

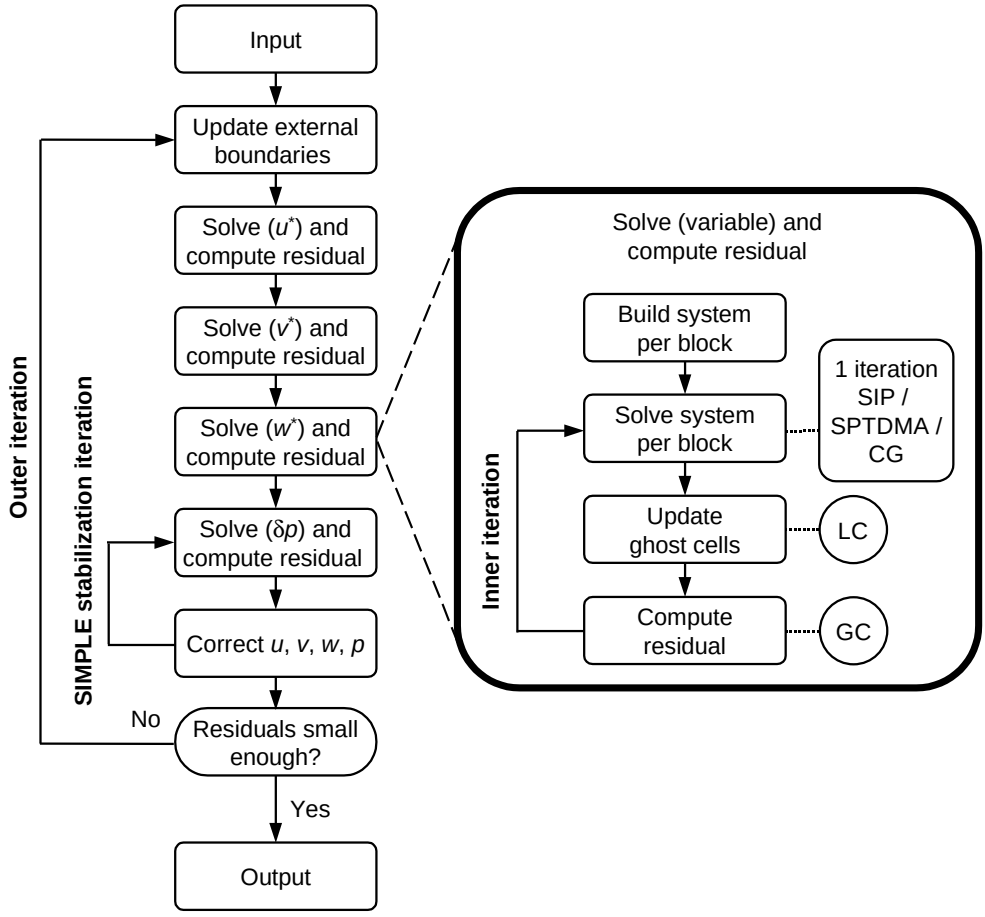

Figure 4.5: Flowchart of the solution procedure in X-stream for solving the stationary incompressible Navier-Stokes equations. (LC: local communication; GC: global communication.)

5

Numerical experiments with domain decomposition for the Poisson equation

5.1 Introduction

In this chapter, the DD methods, as described in the preceding chapter, are applied to the 1D and 2D Poisson equation. The main reason for doing this, is to gain insight into these methods. The numerical experiments are done using the package MATLAB (MATrix LABoratory). This package is used since MATLAB has a huge number of build-in routines for matrix analysis, and programming in MATLAB is elementary. Therefore, we can mainly focus on the behavior of the DD algorithms, and the algorithms can be easily verified for their correctness. All experiments are done on a Pentium III 450MHz machine, having 256 MB of internal memory.

The structure of this chapter is as follows. The setup for the experiments is given in Section 5.2. In Section 5.3 the results are discussed for the Poisson equation with homogeneous Dirichlet BCs, in Section 5.4 the same is done for the case of homogeneous Neumann BCs. Finally, the connection with the X-stream code is given in Section 5.5.

5.2 The 2D Poisson equation and setup

Consider the simple geometry of a unit square for which we have a constant number of subdomains in each dimension (see Subsection 4.3.2). The Poisson equation for this geometry reads:

$$\Delta y(\boldsymbol{x}) = b(\boldsymbol{x}) \ , \quad \boldsymbol{x} \in \Omega \equiv (0,1) \times (0,1) \ . \tag{5.1}$$

For simplicity, we restrict ourselves to a cell-centered equidistant grid with a grid spacing h (see Figure 2.1 for the case $h_1 = h_2$). Consider a finite volume discretization of Equation (5.1) using a 5-point computational molecule:

$$\begin{array}{ccc} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{array} \ . \tag{5.2}$$

Partitioning is done as illustrated in Figure 5.1, for the case of a 2×2 decomposition of a 6×6 grid. The subdomains are numbered from bottom-left to top-right. A global numbering of the grid points for the total domain is used, for which the grid points are numbered from bottom to top in each subdomain.

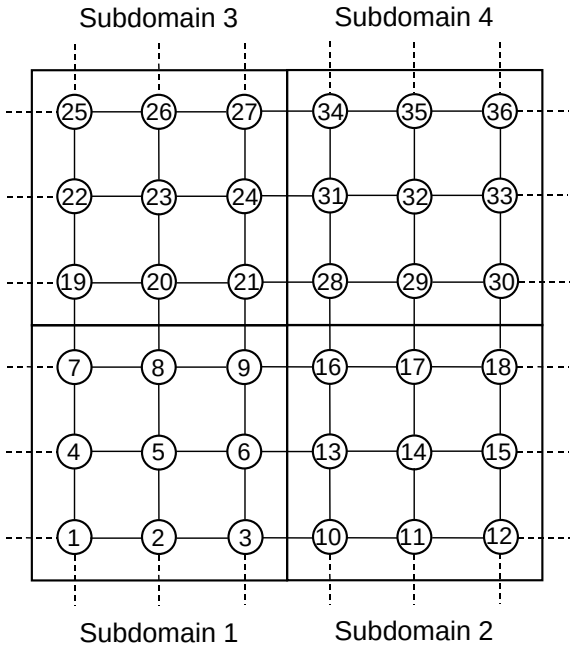

Figure 5.1: Example partitioning of a 2×2 decomposition, and 3×3 grid cells per subdomain.

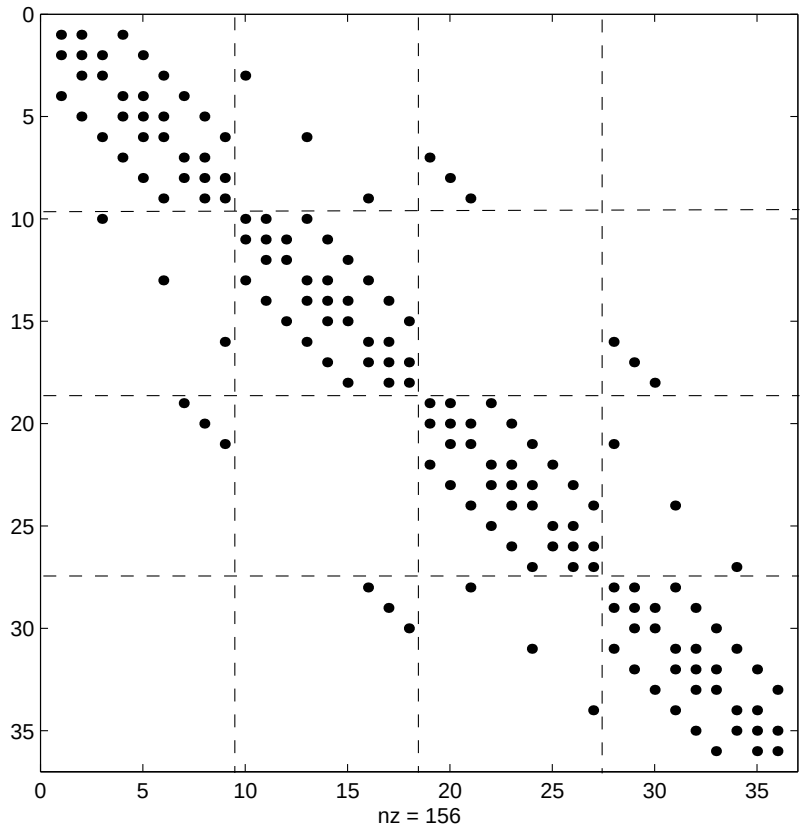

Figure 5.2: Non-zero pattern of the matrix A, for the case of a decomposition and grid as in Figure 5.1.

In this chapter, we consider homogeneous Dirichlet BCs and homogeneous Neumann BCs. For the Dirichlet problem we take

$$y(\boldsymbol{x}) = 0 \ , \qquad \boldsymbol{x} \in \partial\Omega \ , \tag{5.3}$$

and for the Neumann problem

$$\frac{\partial y(\boldsymbol{x})}{\partial n} = 0 \ , \qquad \boldsymbol{x} \in \partial\Omega \ , \tag{5.4}$$

where n denotes the outward unit normal on the boundary.

The Dirichlet BCs are implemented as follows, see Figure 5.1. Consider an unknown with a connection across the (external) boundary, *i.e.*, with a connection to a virtual point (or ghost point). Although we only have a homogeneous Dirichlet condition at the boundary, we take the value in the ghost point equal to the value at the boundary, *i.e.* zero[1]. Therefore, for an unknown near the boundary, the computational molecule is equal to (5.2) without the connections across the boundaries.

For the Neumann condition, we let the value of the ghost point to be equal to the unknown near the boundary. Hence, the computational molecule (5.2) has no connections pointing outward of the boundary, and the central element 4 is then reduced by 1 for each missing connection.

Partitioning and discretization for the example of Figure 5.1, results in a system

$$A\boldsymbol{y} = -h^2\boldsymbol{b} \ , \tag{5.5}$$

with the matrix A, given by

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \emptyset \\ A_{21} & A_{22} & \emptyset & A_{24} \\ A_{31} & \emptyset & A_{33} & A_{34} \\ \emptyset & A_{42} & A_{43} & A_{44} \end{bmatrix} \ .$$

The non-zero pattern of A is shown in Figure 5.2.

In our experiments, we will take $-h^2\boldsymbol{b}$ as a vector with random entries between 0 and 1. We will denote this vector simply by $\boldsymbol{b}$.

5.3 The Dirichlet problem

In Subsection 5.3.1, the choice of deflation vectors is discussed. Next, the results for the eigenvalue spectra and the iterative solution methods are presented in Subsection 5.3.2, followed by some further observations in Subsection 5.3.3.

5.3.1 Choice of deflation vectors

Choosing the deflation vectors for obtaining an efficient iterative solution method is not a trivial task. Several considerations have to be taken into account. For our purpose, we want the matrix Z to satisfy the following properties:

1. The matrix Z should be problem independent and inexpensive to construct.

[1] This treatment of the homogeneous Dirichlet BCs seems to be conform to Frank & Vuik [6].

2. Generalization to the 3D case should be possible, regarding the X-stream code.

3. The matrix Z should be chosen such that $E = Z^T AZ$ is non-singular, because systems involving E need be computed by a direct method. This is especially important for the Neumann problem, see Section 5.4.

4. The span of the columns of Z should approximate the eigenspace belonging to the smallest eigenvalues as good as possible, effecting the rest of the spectrum as little as possible.

5. The gain in iterations for this choice of Z should result in lower wall-clock times.

Property 4 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 proven, see Nicolaides [14] and Frank & Vuik [5]. We will go into more details in Subsection 5.3.3. Property 5 can depend on a number of interacting factors: 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 systems involving E.

The question is: how can we choose Z such that the above properties are satisfied?

Constant deflation vectors combined with linear deflation vectors

Many authors consider constant deflation (CD deflation), for which Z is taken as in (4.16), see Subsection 4.3.2. We will now illustrate CD deflation for the 1D case, which is most suitable for illustrating the basic concept. In the left plot of Figure 5.4, an example is given for the case of two subdomains with each 4 grid cells. For each subdomain, exactly one deflation vector is defined having elements that are constant in the grid points on the corresponding

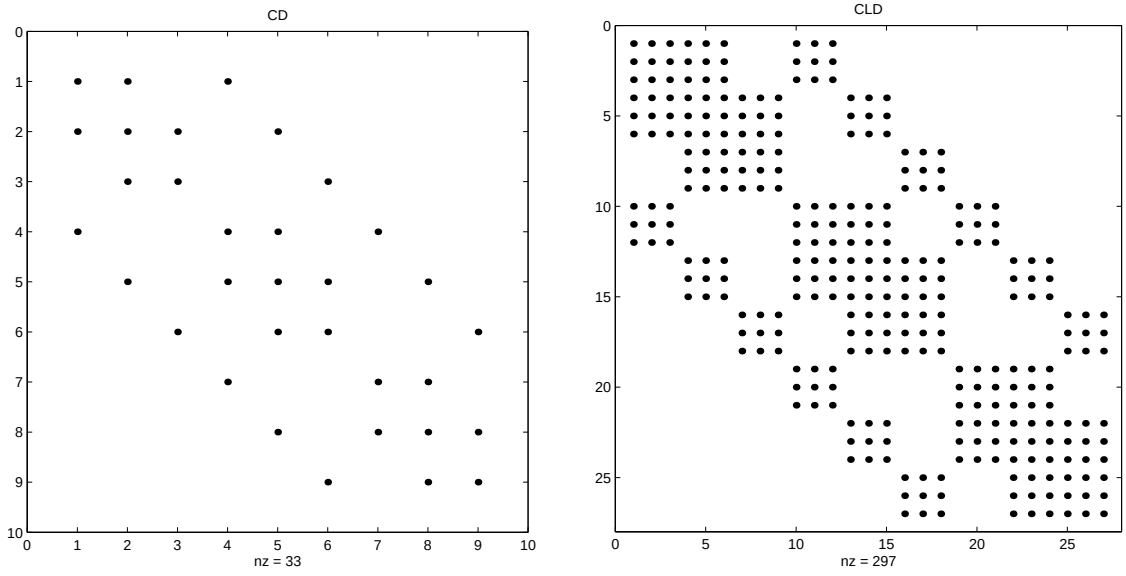

Figure 5.3: Non-zero pattern of E for the 1D case of 3 subdomains. Left: E for CD deflation; right: E for CLD deflation.

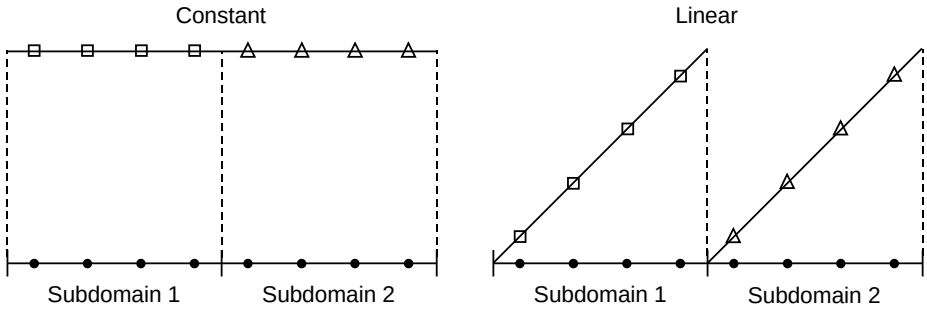

Figure 5.4: Deflation vectors for the 1D example of two subdomains with 4 grid cells per subdomain. Left: CD deflation; right: CLD deflation.

subdomain, and zero elements in the grid points on the other subdomain. For this example, the matrix Z is given by, when the grid points are numbered according to Figure 5.1:

$$Z_{\mathrm{CD}} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}.$$

The value 1 in Z can be replaced by some different non-zero value, since the matrix PA only depends on span$\{Z\}$. In the left plot of Figure 5.3, an example of the non-zero pattern of E is given for the 1D case of 3 subdomains.

Generalization to the more dimensional case is straightforward: for a subdomain, we simply take the elements of the deflation vector to be constant in the grid nodes.

Approximation of an eigenvalue is done as follows. Consider the 1D case for 3 subdomains, then the top plot of Figure 5.5 shows an example of an eigenvector belonging to the smallest eigenvalue, approximated by CD deflation. We see that CD gives a rough approximation of this eigenvector. This motivates us to augment the subspace spanned by the constant deflation vectors with linear deflation vectors. By this, we can approximate the eigenvector as depicted in the bottom plot of Figure 5.5.

Figure 5.4 shows how the linear deflation vectors are defined. A linear deflation vector is defined on each subdomain by increasing each element linearly for increasing grid points. The matrix Z, for which the columns span the space consisting both the constant and the linear vectors, we will denote by constant linear deflation (CLD). For the case of Figure 5.4, this matrix is given by:

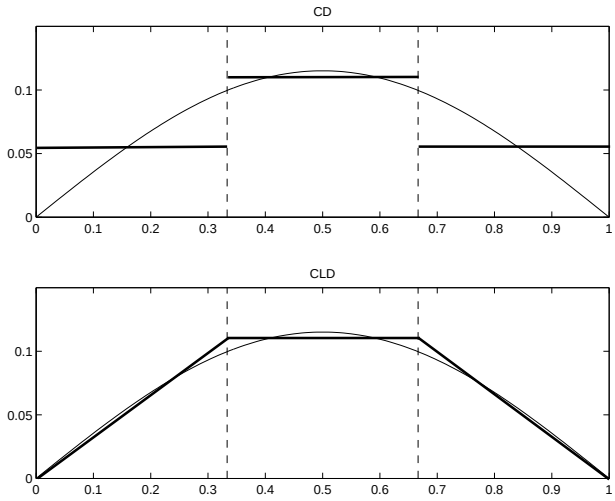

Figure 5.5: Illustration of how the eigenvector belonging to the smallest eigenvalue can be approximated. Top: approximation by constant vectors. Bottom: approximation by linear combinations of constant and linear vectors.

$$Z_{\text{CLD}} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ 1 & 4 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 1 & 4 \end{bmatrix}.$$

Hence, per subdomain we have two deflation vectors: one constant and one linear. Note that the values of the linear deflation vectors does not matter as long as the vector is linear, since only span$\{Z\}$ counts. In the right plot of Figure 5.3, an example of the non-zero pattern of E is given for the 1D case of 3 subdomains.

Generalization to 2D and 3D is not so straightforward anymore, compared to CD deflation. However, it appears that in two dimensions Z should consists of 3 vectors: one constant, and one linear vector in each of the two dimensions. For the 3D case, it appears that we need 4 deflation vectors: one constant and one linear vector in each of the three dimensions.

In the following subsection, we will compare CD deflation with CLD deflation for the 2D case. First, a short comparison of the eigenvalues is given. To illustrate that the choice of Z is not trivial, we will also compare CLD deflation with another choice for Z.

How this Z is constructed, is illustrated by Figure 5.6 for the 1D case. For this example, the constant deflation vector is splitted into two separate vectors for each subdomain. By this, an eigenvector can be approximated as depicted in Figure 5.7. For the 2D case we will split the constant deflation vector into three vectors, in order to obtain the same number of deflation vectors as for CLD deflation. For brevity, we will refer to this choice of Z as 3CD deflation.

Secondly, we will compare the additive Schwarz method, the GCR-Schwarz method and

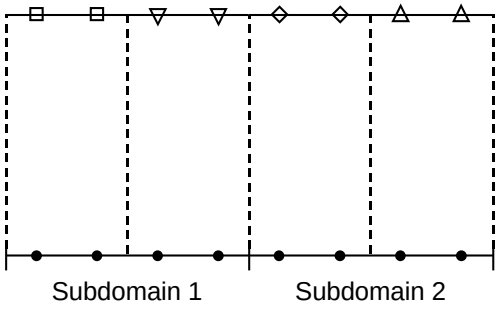

Figure 5.6: Two constant deflation vectors per subdomain, for the 1D case of two subdomains and 4 grid cells per subdomain.

the CG-Schwarz method, as discussed in Chapter 4, for a fixed problem size and a varying number of decompositions.

5.3.2 Eigenvalue spectra and iterative solution methods compared

Consider the case of a 3×3 decomposition of a 9×9 grid. Consider the matrix $P = I - AZE^{-1}Z^T$ for the case of CD and CLD deflation, denoted by P_{CD} respectively P_{CLD}. Let the eigenvalues for this example be ordered as $\lambda_1 \leq \cdots \leq \lambda_{81}$. Then the eigenvalue spectra of the $M_{\text{jac}}^{-1}A$, $M_{\text{jac}}^{-1}P_{\text{CD}}A$, and $M_{\text{jac}}^{-1}P_{\text{CLD}}A$ are given by Figure 5.8, were M_{jac} is the block Gauß-Jacobi preconditioner.

We observe the expected clustering around 1, as mentioned in Subsection 3.2.1. Furthemore, we see that $\lambda_1, \ldots, \lambda_9$ are set to zero for CD deflation, and $\lambda_1, \ldots, \lambda_{27}$ for CLD deflation. This has the effect that several eigenvalues of $M_{\text{jac}}^{-1}A$ are changed. We observe that the smallest non-zero eigenvalue of $M_{\text{jac}}^{-1}P_{\text{CLD}}A$ is larger than λ_1, and that the largest eigenvalue is smaller than λ_{81}. This means that the effective condition number of $M_{\text{jac}}^{-1}P_{\text{CLD}}A$ is the smallest of the two deflation choices, and is therefore the most favorable. Figure D.1 of Appendix D.1 shows the three different spectra for the unpreconditioned case.

The spectra for CLD deflation is compared to 3CD deflation in Figure 5.9. Clearly, we see that 3CD deflation results into a less favorable spectrum compared to CLD deflation. For this reason, we will restrict ourselves entirely to CD and CLD deflation.

Now consider a small fixed problem size of a 12×12 grid. The effective condition numbers,

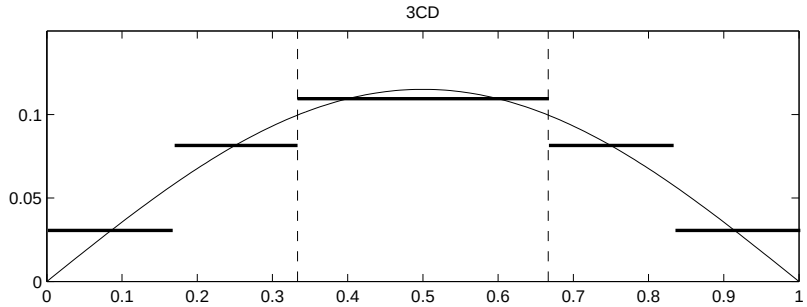

Figure 5.7: Illustration of how the eigenvector belonging to the smallest eigenvalue can be approximated using two constant deflation vectors per subdomain.

for three decompositions, are given by the following table:

κ_{eff}	2×2	3×3	4×4
A	67.83	67.83	67.83
$M_{\text{jac}}^{-1}A$	13.00	17.94	23.29
$P_{\text{CD}}A$	25.51	12.61	7.45
$M_{\text{jac}}^{-1}P_{\text{CD}}A$	3.62	2.91	2.46
$P_{\text{CLD}}A$	9.66	5.56	3.51
$M_{\text{jac}}^{-1}P_{\text{CLD}}A$	2.23	1.97	1.84

Table 5.1: Effective condition numbers for three decompositions. The Dirichlet problem is considered and a 12×12 grid is taken.

From this table we can conclude the following. As expected, the condition number of A is independent on the number of subdomains, because the condition number is independent on the ordering used. We see the the preconditioning of A becomes worse for an increasing number of subdomains. This is because M_{jac} does not contain the coupling matrices, for which the number increases for an increasing number of subdomains.

Furthermore, we see that the condition numbers for CLD deflation are better compared to CD deflation, illustrating our previous observation. We observe that the effective condition number of $M_{\text{jac}}^{-1}P_{\text{CD}}A$ and $M_{\text{jac}}^{-1}P_{\text{CLD}}A$ slightly improves for an increasing number of subdomains. From this, we expect that the deflated GCR-Schwarz and CG-Schwarz are well scalable. Furthermore, the difference between these conditions numbers decreases for an increasing number of subdomains. Hence, we expect that CD deflation and CLD deflation results in a more similar performance for an increasing number of subdomains.

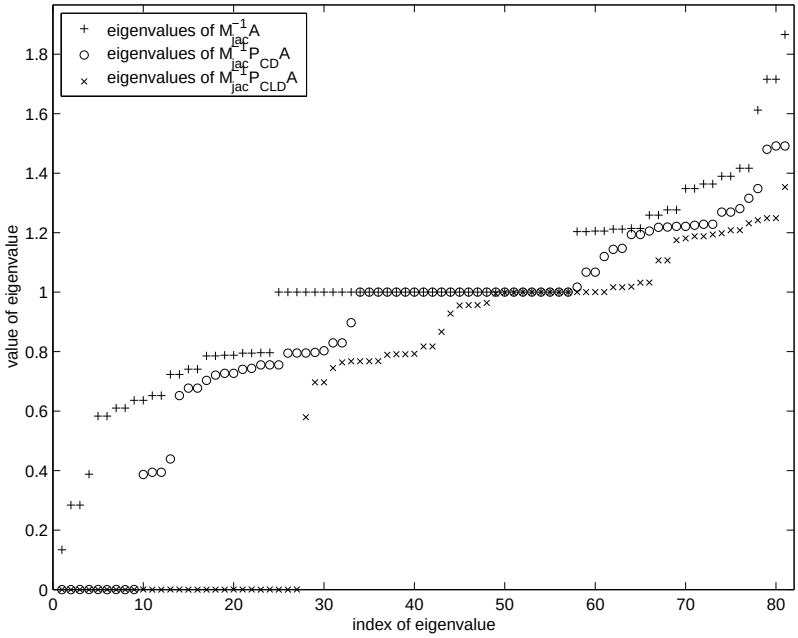

Figure 5.8: Eigenvalue spectra for the preconditioned case, for the case of no deflation, CD deflation and CLD deflation. The Dirichlet problem is considered for a 9×9 grid and a 3×3 decomposition.

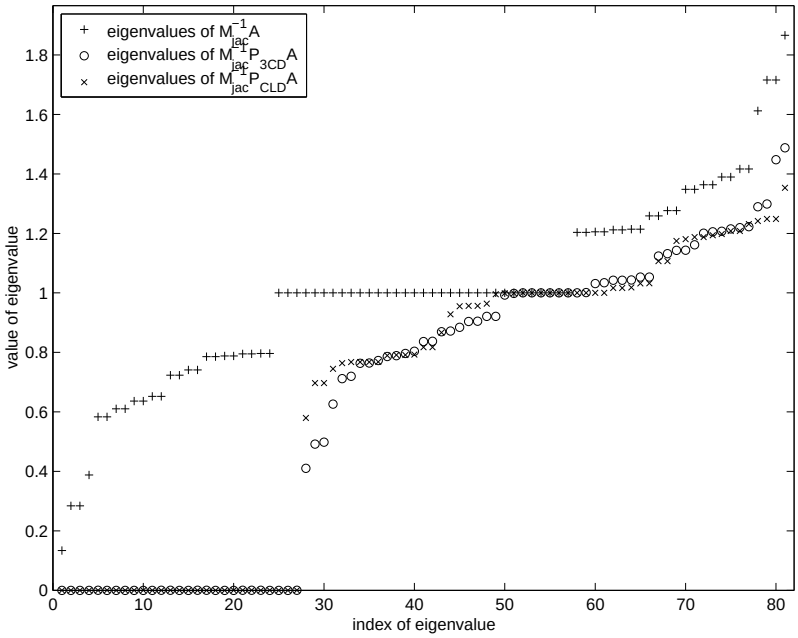

Figure 5.9: Eigenvalue spectra for the preconditioned case, for the case of no deflation, CLD deflation and 3CD deflation. The Dirichlet problem is considered for a 9×9 grid and a 3×3 decomposition.

Iterative solution methods compared for a fixed problem size

In the preceding, we concluded that deflation reduces the effective condition numbers considerably. We will now investigate what influence this effect has on the Deflated Krylov-Schwarz methods. The methods we will compare are the GCR-Schwarz method, the CG-Schwarz method and the additive unaccelerated Schwarz method. These three methods are combined with CD and CLD deflation, resulting in, respectively, Algorithm 4.1, Algorithm 4.2, and Algorithm 3.4.

The subdomain problems are solved using a BIM with a standard ILU preconditioning. A fixed number of iterations are performed for solving the subproblems: 15, 5 or 2, see Subsection 4.3.3. Restarted GCR and truncated GCR are considered for 5 search directions. For example, with no deflation we denote this algorithms by respectively GCR(5r) and GCR(5t). Furthermore, MGS orthonormalization is applied. The unaccelerated Schwarz method will be denoted by simply ILU. A zero starting vector is used for all methods as well as for obtaining the subdomain solution. Furthermore, an absolute termination criterion (3.10) for a tolerance of 10^{-6} is taken. The maximum number of iterations was set to 250 iterations. We consider a 60×60 grid and four different decompositions: 2×2, 3×3, 4×4 and 5×5.

In Table D.1 in Appendix D.2, the Schwarz iterations, as well as the wall-clock time (in parentheses) are given. Figure 5.12 shows the scalability results for the GCR-Schwarz, GCR(5t)-Schwarz, CG-Schwarz and ILU method, for the case of two subdomain iterations. The ILU method without deflation is not shown in this figure, since this method does not converge in less than 250 iterations. Figures 5.10 and 5.11, respectively, show the converge behavior for respectively the Krylov-Schwarz methods and the ILU method, for the case of 2 subdomain

iterations taken.

From Figure 5.10 and Table D.1, we conclude we the GCR-Schwarz and CG-Schwarz methods perform quite similarly. Furthermore, we see that CLD deflation results in the best convergence behavior. Noticeable is that the ILU method performs very poorly compared to GCR and CG. On the other hand, the CDILU method performs very well for the case of 16 and 25 subdomains. From Figure 5.12, we conclude that the Deflated Krylov-Schwarz methods are well scalable, and that the convergence rate is more or less independent on the number of subdomains. This is conform our expectations from Table 5.1.

Concerning wall-clock times, we observe that the wall-clock time for the GCR-Schwarz method is minimal for a 4×4 decomposition. Likely, there are optimal decompositions for the deflation methods such that the wall-clock time is minimal. Furthermore, we observe that much time is gained by solving the subdomain inaccurately, and that the number of iterations does not significantly increases for the case of 9, 16 and 25 subdomains. From this, we conclude that we solve the subdomain solutions too inaccurate with 2 iterations, for the case of 4 subdomains. Note, that the wall-clock times are strange when we compare, for example, the timings for 4 subdomains and 9 subdomains. Doing the same number of iterations requires approximately a factor 2 to 4 more wall-clock time. A possible explanation for this behavior is the implementation of the couplings, for which elements in the corresponding arrays need to be searched. This costs a lot of 'if' statements and 'for' loops, which are very expensive in MATLAB.

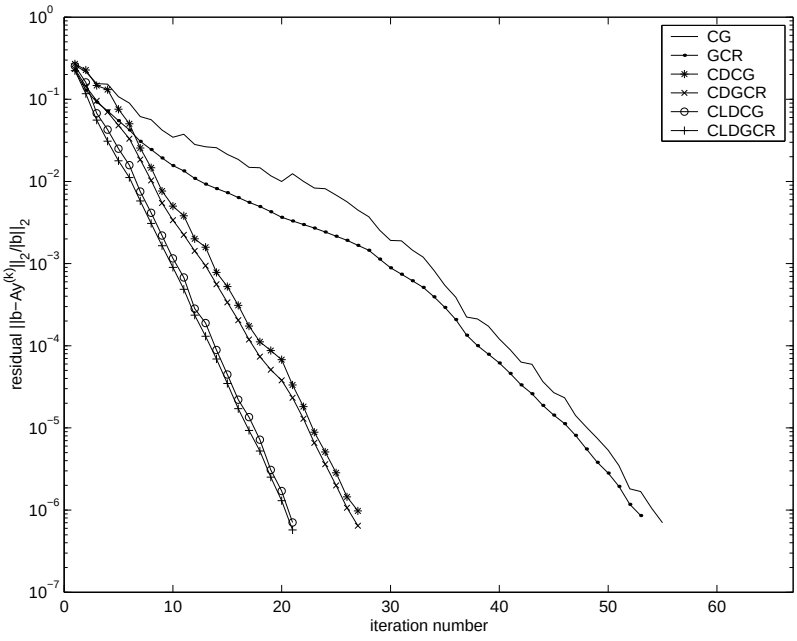

Figure 5.10: Residuals for the GCR-Schwarz and CG-Schwarz methods considering no deflation, CD deflation, and CLD deflation. The results are for the Dirichlet case and for a 5×5 decomposition of a 60×60 grid. Two iterations for solving the subdomain problems are taken.

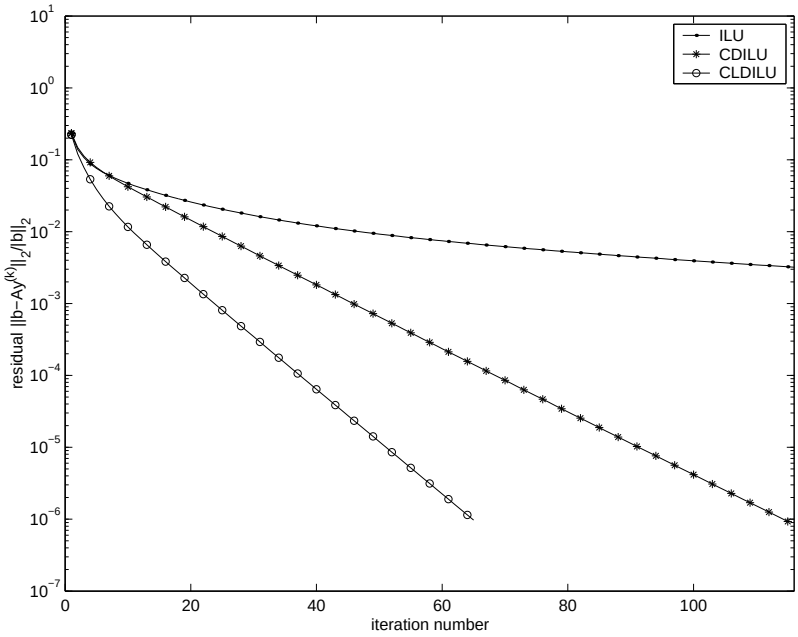

Figure 5.11: Residuals for the ILU method considering no deflation, CD deflation, and CLD deflation. The results are for the Dirichlet case and for a 5×5 decomposition of a 60×60 grid. Two iterations for solving the subdomain problems are taken.

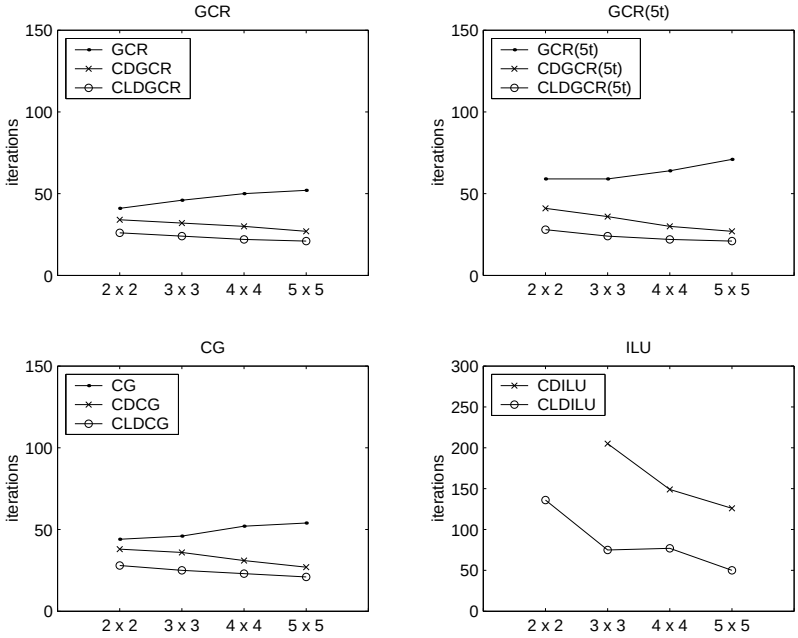

Figure 5.12: Scalability results for the solution methods for the Dirichlet problem.

The additional work and storage due to deflation

Deflation introduces additional work and storage, see for example Algorithm 4.1. We will now give a rough estimation for the overhead deflation introduces.

Let d be denoted as the spatial dimension, as usual. In general, DD is implemented as follows. A matrix A_{ii}, representing the discretization on a subdomain i, is usually very sparse and therefore only the diagonals $(2d+1)$ are stored. Because of the sparsity of the coupling matrices A_{ij}, $i \neq j$, only the non-zero elements are stored in an array. A subdomain topology data structure is used to connect the subdomains, see Brakkee [2, Sect. C.1.2] for an example.

We restrict ourselves to a domain which has the unit length in each dimension. Furthermore we consider a uniform grid. Consider a decomposition of M subdomains in each dimension of equal size, thus consisting in total of M^d subdomains. Let n be the number of grid cells each subdomain has in one dimension, hence n^d is the number of grid cells per subdomain. Furthermore, let m be the number of deflation vectors per subdomain, for which only the non-zero elements are stored in an array. For simplicity, consider the case that a full LU factorization is applied to factorize $E = Z^T AZ$, and that the result is stored in a matrix L and a matrix U. We will neglect matrix vector multiplications involving coupling matrices, and we roughly assume that a multiplication of A_{ii} with a vector takes $(4d+2)n^d$ floating point operations (flops), ignoring boundary effects. The number of flops for the LU factorization, the LU forward– and backward–solution, can be found in, for example, Golub & Van Loan [7, page 152]. By this, the extra work (in flops) and storage (in memory positions) due to deflation, can be summarized by the following table:

Statements	*Work*	*Storage*
Construct Z		mM^dn^d
Compute AZ	$(4d+2)mM^dn^d$	mM^dn^d
Compute E	$2mM^dn^d$	m^2M^{2d}
LU factorization of E	$\frac{2}{3}m^3M^{3d}$	$2m^2M^{2d}$
Solve $E\boldsymbol{q}_1 = Z^T\boldsymbol{r}^{(0)}$	$2m^2M^{2d}$	mM^d
$\boldsymbol{r}^{(0)} := \boldsymbol{r}^{(0)} - AZ\boldsymbol{q}_1$	$(2m+1)M^dn^d$	
Solve $E\boldsymbol{q}_2 = Z^T\boldsymbol{v}$	$2m^2M^{2d}$	mM^d
$\boldsymbol{v} := \boldsymbol{v} - AZ\boldsymbol{q}_2$	$(2m+1)M^dn^d$	
Compute $A\boldsymbol{y}^{(k)}$	$(4d+2)M^dn^d$	M^dn^d
Solve $E\boldsymbol{q}_2 = Z^TA\boldsymbol{y}^{(k)}$	$2m^2M^{2d}$	mM^d
$\boldsymbol{y} := \boldsymbol{y} + \boldsymbol{y}^{(0)} + Z(\boldsymbol{q}_1 - \boldsymbol{q}_2)$	$(2m+2)M^dn^d$	

Table 5.2: A rough estimation for the additional work and storage deflation introduces.

From this table we conclude that for $n \gg M$, CLD deflation roughly costs three times as much as work for the 2D case than CD deflation, and four times as much in the 3D case. We also conclude that for the 3D case, terms with M^{2d} and M^{3d} could become significantly large for a large number of subdomains. Therefore, work and storage for the solving the systems involving E could become expensive. One way to decrease computing time is to use a band / profile LU factorization of E, when the subdomain numbering is structured. A way to reduce storage is to simply overwrite the matrix A with the LU factorization.

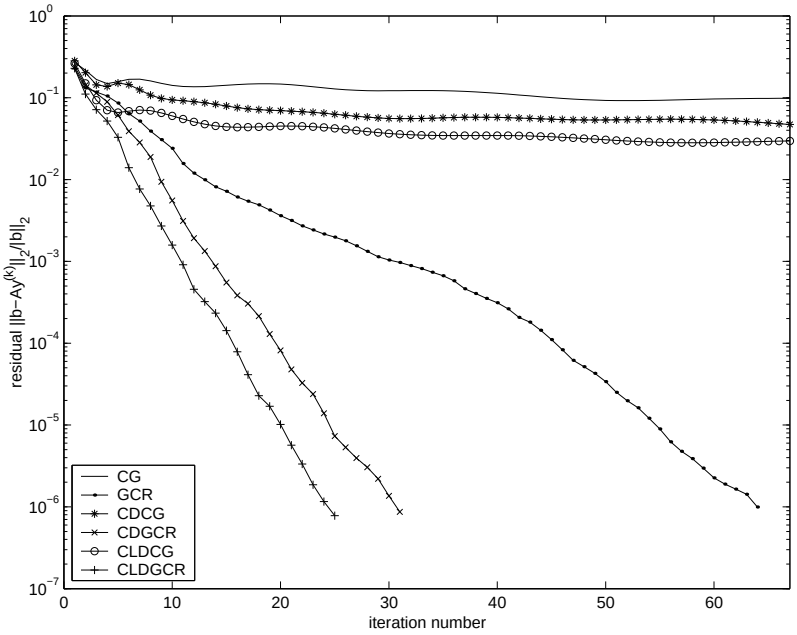

Figure 5.13: The effect of choosing an old search direction as a starting vector for solving the subdomain problems. The results are for the Dirichlet case and for a 5×5 decomposition of a 60×60 grid. Two iterations for solving the subdomain problems are taken.

5.3.3 Further observations

Choice of starting vector for solving the subdomain problems

In Subsection 3.2.1, it was mentioned that when no further information is available for choosing a starting vector, it is advisable to choose the zero vector. However, for solving the search directions in the GCR-Schwarz and CG-Schwarz, one could think that such information is available, *i.e.* the (orthonormalized) search directions of the previous iteration. Consider the case of a 5×5 decomposition of a 60×60 grid, and let the subdomain problems be solved by two iterations. Taking the starting vector equal to the old search direction, this results in a convergence behavior as depicted in Figure 5.13. We observe that the GCR-Schwarz method performs less good compared to the case of a zero starting vector (see Figure 5.10), while the CG-Schwarz method does not converge at all. From this observation, we conclude that a zero starting vector is most preferable for solving the subdomain problems.

A splitting of A for the 1D case and CLD deflation

In Frank & Vuik [5], the following theorem is proven, which gives us a bound for the spectrum of PA:

Theorem 5.1. *Let A be a symmetric positive definite matrix, P defined by (3.24), and suppose there exists a splitting $A = M + N$ such that M and N are symmetric positive semidefinite with* $\text{Null}(M) = \text{span}\{Z\}$ *the null space of M. Then*

$$\lambda_i(M) \leq \lambda_i(PA) \leq \lambda_i(M) + \lambda_{\max}(PN) \ . \tag{5.6}$$

Moreover, the effective condition number of PA is bounded by

$$\kappa_{\text{eff}}(PA) \leq \frac{\lambda_n(A)}{\lambda_{m+1}(M)} \ . \tag{5.7}$$

Proof. See Theorem 2.2 in Vuik & Frank [24]. □

This theorem can give us important insight into the interpretation of the deflation method. In Vuik & Frank [24], an example is given for a splitting of a matrix A, where A is the matrix resulting from the discretization of the Dirichlet problem. Let $\Sigma(M_{\text{jac}})$ be denoted as the vector containing the row sums of M_{jac}, the block Gauß-Jacobi preconditioner. Then the following splitting satisfies the conditions of Theorem 5.1 for the case of CD deflation:

$$A = M + N \ , \quad M \equiv M_{\text{jac}} - \Sigma(M_{\text{jac}}) \ . \tag{5.8}$$

A block of M_{ii} can be interpreted as a discretization for a Neumann problem on a subdomain. With (5.6) and (5.7), this splitting tells us that CD effectively decouples the original system into a set of independent Neumann problems on the subdomains, with convergence governed by the 'worst' conditioned Neumann problem.

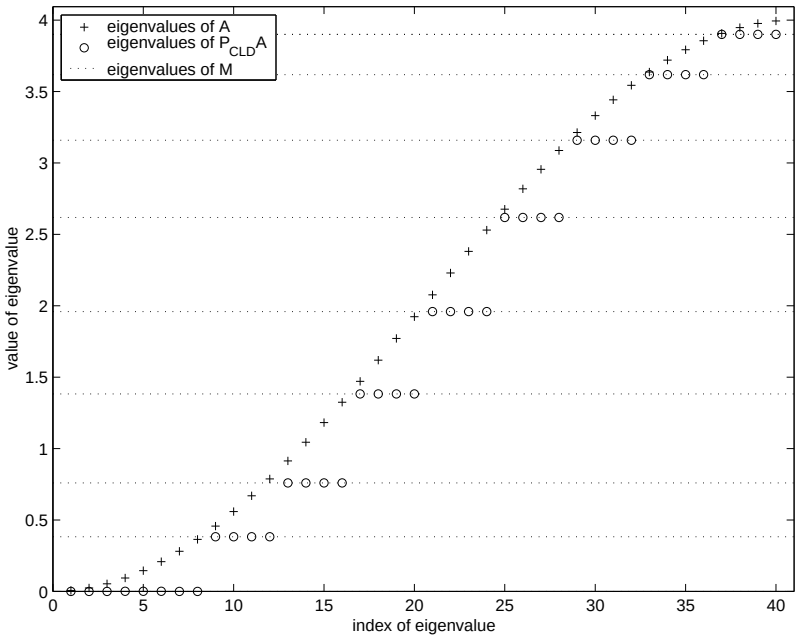

Figure 5.14: Eigenvalue spectra for the splitting for the case of CLD deflation. The 1D Dirichlet problem is considered for 4 subdomains and 10 grid cells per subdomain, and the unpreconditioned case is considered.

The question one could ask now is: can we also find a suitable splitting for the case of CLD deflation that satisfies the conditions of Theorem 5.1? For the 1D case this seems possible, and the blocks M_{ii} of M could be chosen as:

$$M_{ii} = \begin{bmatrix} \frac{n-2}{n-1} & -1 & 0 & \cdots & 0 & \frac{1}{n-1} \\ -1 & 2 & -1 & & & 0 \\ 0 & -1 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & -1 & 0 \\ 0 & & & -1 & 2 & -1 \\ \frac{1}{n-1} & 0 & \cdots & 0 & -1 & \frac{n-2}{n-1} \end{bmatrix},$$

where n is the number of grid cells per subdomain. Numerical experiments indicate that the conditions of Theorem 5.1 are satisfied for this splitting. Furthermore, they indicate that $PN = 0$, and therefore Equation (5.6) can be replaced by $\lambda_i(PA) = \lambda_i(M)$. From this, it follows that, $\kappa_{\text{eff}}(PA) = \kappa_{\text{eff}}(M)$.

Figure 5.14 shows an example for the case of 4 subdomains, each containing 10 grid cells. The eigenvalue spectra of A, PA and M are given, illustrating our observations.

Numerical experiments using M_{jac} also indicate that the preconditioned version of Theorem 5.1 holds, as given in Frank & Vuik [5, Theorem 2.3]. Figure D.3 in Appendix D.1 gives an illustration for our example.

A proof and an interpretation for all these observations do not seem straightforward, and also a generalization to the 2D and 3D case is not obvious. Therefore, this is left for further research.

5.4 The Neumann problem

In Subsection 5.4.1, first the choice of deflation vectors is discussed. Then the results for the eigenvalue spectra and the iterative solution methods are presented in Subsection 5.4.2. Finally, some further observations with MATLAB are discussed in Subsection 5.4.3.

5.4.1 Choice of deflation vectors

When homogeneous Neumann BCs are chosen, the matrix A turns out to be singular. The solution of the system $A\boldsymbol{y} = \boldsymbol{b}$ is determined up to a constant, and the vector with all 1's is an eigenvector of A with eigenvalue 0. The system has a solution if, and only if, the vector $\boldsymbol{b} \in \text{ran}(A)$. When this is the case, then the system is called consistent. Two methods can be distinguished to solve a non-consistent system, see Kaasschieter [9] and van der Vorst [34].

The first method is to fix one entry of $\boldsymbol{y}$, deleting the corresponding row and column of A, adjusting $\boldsymbol{b}$ and solving the resulting non-singular system. However, it turns out that this can have an infavorable effect on the convergence behavior of the solution method used.

A better approach is to adjust $\boldsymbol{b}$ by projecting $\boldsymbol{b}$ onto $\text{ran}(A)$ and make it orthogonal to the eigenvector corresponding to the zero eigenvalue:

$$\boldsymbol{b} \equiv \boldsymbol{b} - (\boldsymbol{b}, \boldsymbol{e})\boldsymbol{e}\ , \tag{5.9}$$

where $\boldsymbol{e}$ is the relevant eigenvector, such that $\|\boldsymbol{e}\|_2 = 1$.

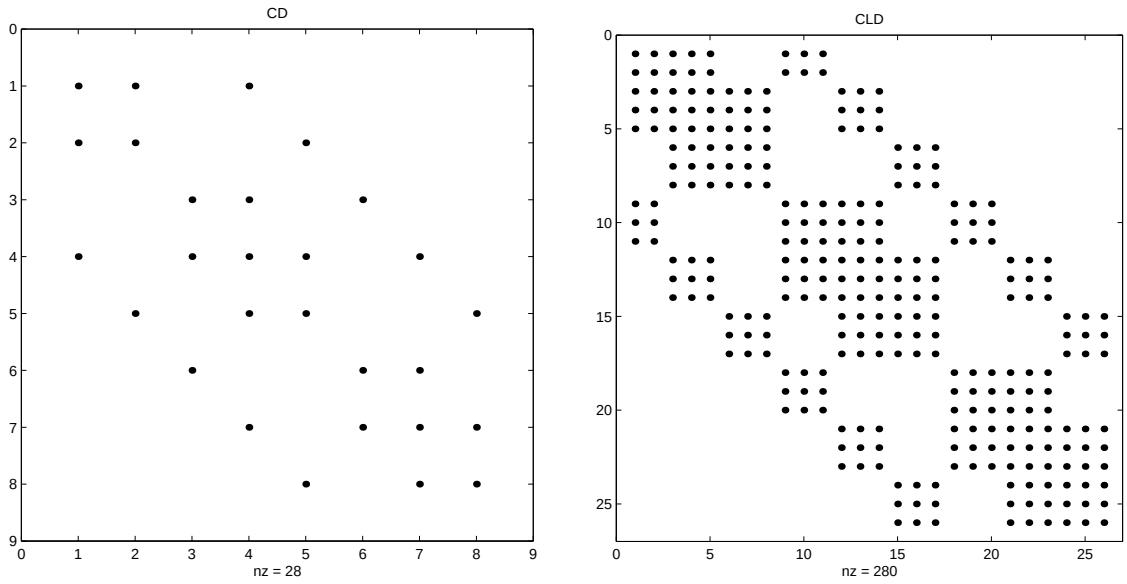

Figure 5.15: Non-zero pattern of E for CD (left) and CLD (right) for the 1D case of three subdomains, when the first constant deflation vector is removed.

The singularity of A has some consequences for our choice of the matrix Z. When Z is chosen for CD and CLD deflation as in Subsection 5.3.1, we have $\boldsymbol{e} \in \text{span}(Z)$, and therefore the matrix $E = Z^T AZ$ is singular. This introduces a problem, when solving the systems involving E with a direct solution method. Therefore, we have to modify Z such that E is non-singular. We will consider two options to overcome this difficulty:

1. Removing a constant deflation vector for one subdomain.

2. Adjusting one entry in a constant deflation vector for one subdomain.

Consider the case of 2 subdomains and 4 grid cells per subdomain. For the first option we take Z for the CD and CLD case as:

$$Z_{\text{CD}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad Z_{\text{CLD}} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \\ 4 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 1 & 3 \\ 0 & 1 & 4 \end{bmatrix}. \tag{5.10}$$

For a 1D example of three subdomains, the non-zero pattern of the matrix E for CD and CLD is given by Figure 5.15. This figure shows that this choice of Z results in an E for which the first row and column are canceled, compared to the Dirichlet case.

For the second option, one can take, for example:

$$Z_{\text{CD}} = \begin{bmatrix} 1+\varepsilon & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \quad Z_{\text{CLD}} = \begin{bmatrix} 1+\varepsilon & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ 1 & 4 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 1 & 4 \end{bmatrix}, \tag{5.11}$$

with ε some number.

Consider a 60×60 grid consisting of 5×5 subdomains, and take Z analogous to (5.11) with $\varepsilon = 10^{-6}$. Then Figure 5.16 shows the convergence behavior of the GCR-Schwarz and the CG-Schwarz methods with CD and CLD deflation. We observe a strange jump in the residuals. For this example, the condition number of E is equal to $1.10 \cdot 10^{15}$ and $1.14 \cdot 10^{17}$ for the CD and CLD case, respectively. Numerical experiments show, that from such large values of $\kappa(E)$, strange convergence behavior occurs.

In Figure 5.17, the influence of $\kappa(E)$ is illustrated for adjustments 1 and 2 of Z, varying ε from -50 to $+50$. We observe that the graphs are more or less symmetric. A peak occurs at $\varepsilon = 0$, and in a small bandwidth around this value $\kappa(E)$ is most sensitive. When we compare CD deflation to CLD deflation, we see that $\kappa(E)$ for CLD deflation is larger for both adjustments of Z. Furthermore, $\kappa(E)$ for CLD deflation seems less sensitive for a decreasing number of grid cells per subdomain. Comparing both adjustment of Z, we see that the asymptotes of the functions in ε more or less coincide with $\kappa(E)$, for the case of removing the first constant deflation vector.

From this simple sensitivity analysis, we conclude that solving systems involving E can be done by removing the first deflation vector of Z, or by adjusting the first entry in the constant deflation vector such that $|\varepsilon|$ is large. Furthermore, we conclude that CLD is more sensitive to the second option. The question one could ask is: what influence do both adjustments of Z have on the performance of the deflation method? Numerical experiments show that the performance is similar, *i.e.*, when ε is chosen such that $\kappa(E) \ll 10^{16}$, the convergence rate for both adjustments is similar.

Other questions that remain are: what and how many constant deflation vector should we remove from Z? What and how many entries of Z should we modify? And, perhaps, what is the behavior concerning rounding errors? These questions are left for further research. From now on, we will choose the option to leave the first deflation vector, since this seems to be the safest option.

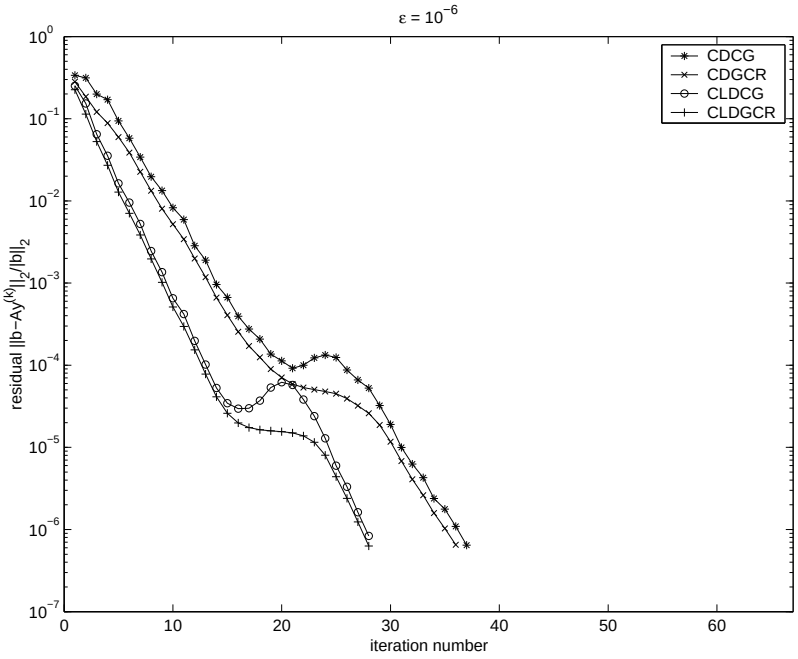

Figure 5.16: Convergence behavior of the GCR-Schwarz and CG-Schwarz methods with CD and CLD deflation for the Neumann problem. A 60×60 grid consisting of 5×5 subdomains is considered, and one entry of the first constant deflation vector of Z is set to $1 + 10^{-6}$.

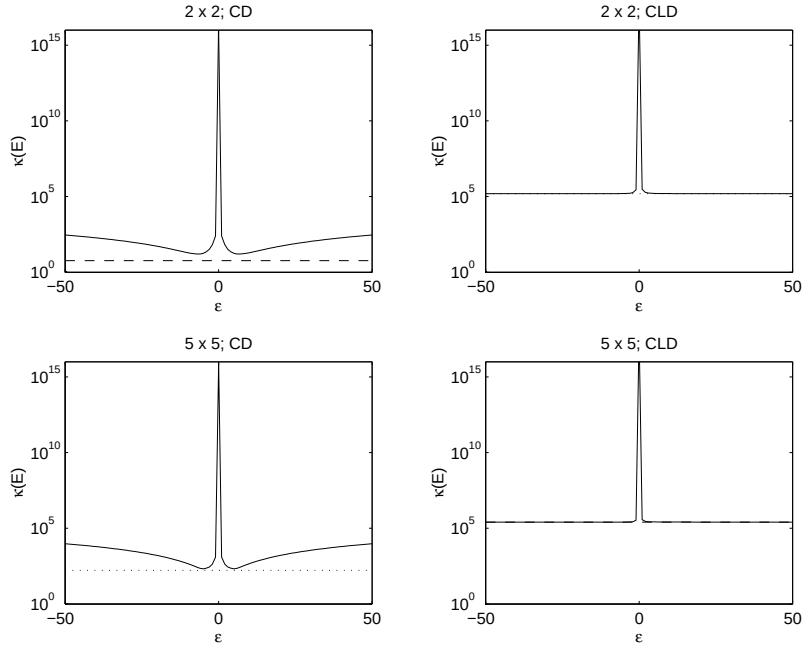

Figure 5.17: Sensitivity of the choice of Z on $\kappa(E)$ for the Neumann problem, when we replace one entry of the constant deflation vector by $1+\varepsilon$. A '$\cdots$' denotes $\kappa(E)$ for the case that the first constant deflation vector is removed.

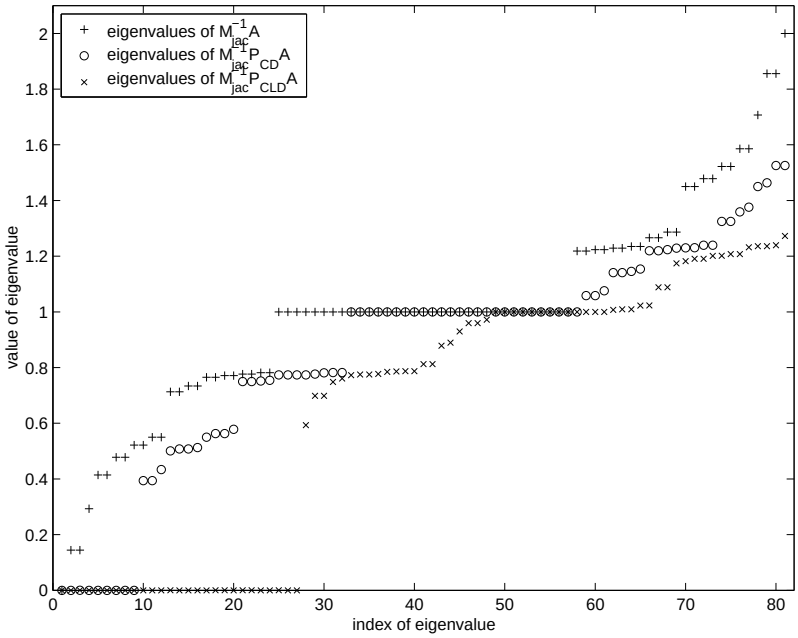

Figure 5.18: Eigenvalue spectra for the preconditioned case, for the case of no deflation, CD deflation and CLD deflation. The Neumann problem is considered for a 9×9 grid and a 3×3 decomposition.

5.4.2 Eigenvalue spectra and iterative solution methods compared

Eigenvalue spectra compared

The eigenvalue spectra for $M_{\text{jac}}^{-1}A$, $M_{\text{jac}}^{-1}P_{\text{CD}}A$, and $M_{\text{jac}}^{-1}P_{\text{CLD}}A$ for the preconditioned case and a 3×3 decomposition of a 9×9 grid are given by Figure 5.18. Figure D.4 in Appendix D.2 shows the spectra for the unpreconditioned case. Note that the first eigenvalue of A is zero. Furthermore, we observe that the spectra are quite similar compared to the Dirichlet problem, see Figure 5.8.

Now consider the case of a 12×12 grid for three different decompositions, just as for the Dirichlet case. The effective condition numbers for $M_{\text{jac}}^{-1}A$, $M_{\text{jac}}^{-1}P_{\text{CD}}A$, and $M_{\text{jac}}^{-1}P_{\text{CLD}}A$ are given by the following table:

κ_{eff}	2×2	3×3	4×4
A	115.39	115.39	115.39
$M_{\text{jac}}^{-1}A$	9.33	18.24	27.22
$P_{\text{CD}}A$	29.31	13.39	7.73
$M_{\text{jac}}^{-1}P_{\text{CD}}A$	6.26	4.76	3.97
$P_{\text{CLD}}A$	14.62	6.63	3.85
$M_{\text{jac}}^{-1}P_{\text{CLD}}A$	2.37	2.30	2.02

Table 5.3: Effective condition numbers for three decompositions. The Neumann problem is considered and a 12×12 grid is taken.

Compared to Table 5.1, we see that the matrices all have a larger condition number. However, the same conclusions as for Table 5.1 can be made.

Iterative solution methods compared for a fixed problem size

For the following results, the random RHS vector $\boldsymbol{b}$ is made consistent according to (5.9). As a termination criterion, the absolute termination criterion is taken for solving the deflated system. Table D.2 shows the results for the fixed problem size, analogous to Table D.1 for the Dirichlet case. Figure 5.21 shows the corresponding scalability results. Compared to the Dirichlet case, the same conclusions hold, although in general more iterations are needed.

Figure 5.19 shows the convergence behavior of the GCR-Schwarz and CG-Schwarz methods, for the case of no deflation, CD deflation, and CLD deflation. Figure 5.20 does the same for the ILU method. Compared to Figures 5.10 and 5.11, we see comparable convergence behaviors.

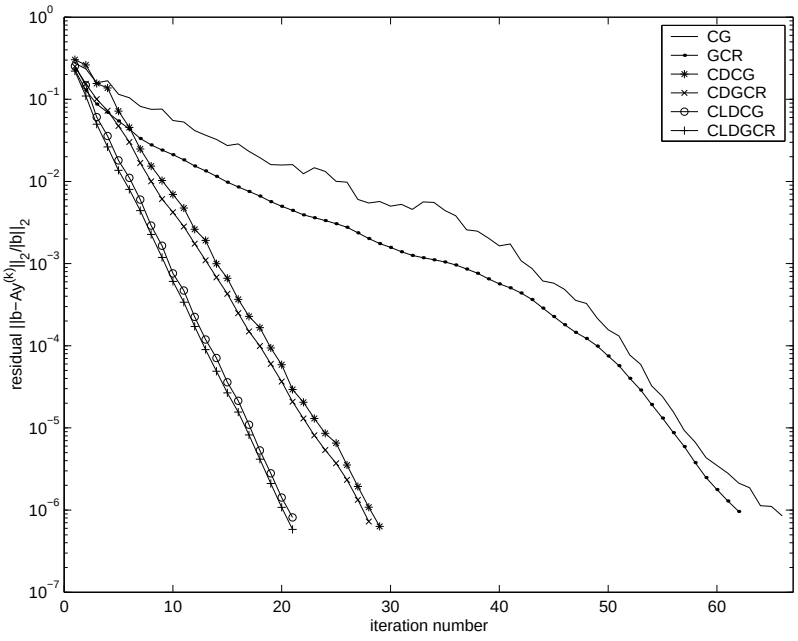

Figure 5.19: Residuals for the GCR-Schwarz and CG-Schwarz methods considering no deflation, CD deflation, and CLD deflation. The results are for the Neumann case and for a 5×5 decomposition of a 60×60 grid. Two iterations for solving the subdomain problems are taken.

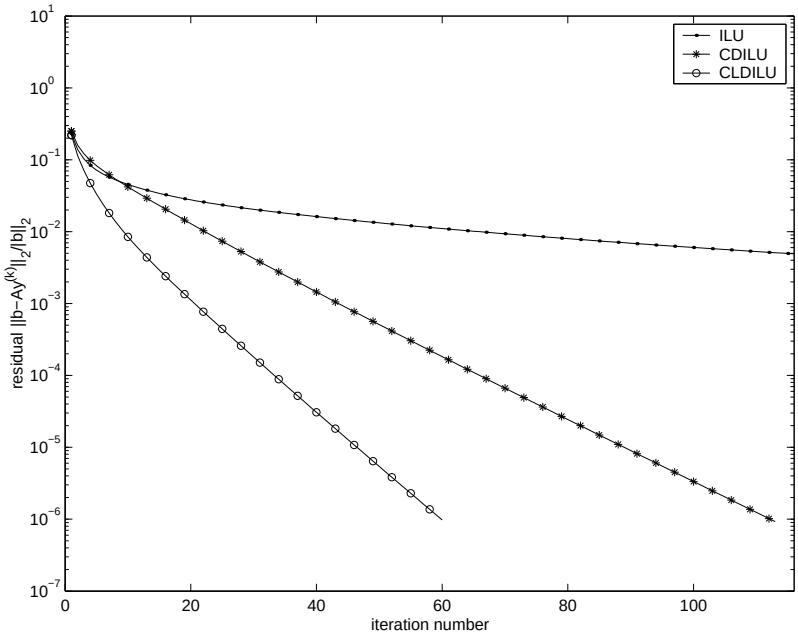

Figure 5.20: Residuals for the ILU method considering no deflation, CD deflation, and CLD deflation. The results are for the Neumann case and for a 5×5 decomposition of a 60×60 grid. Two iterations for solving the subdomain problems are taken.

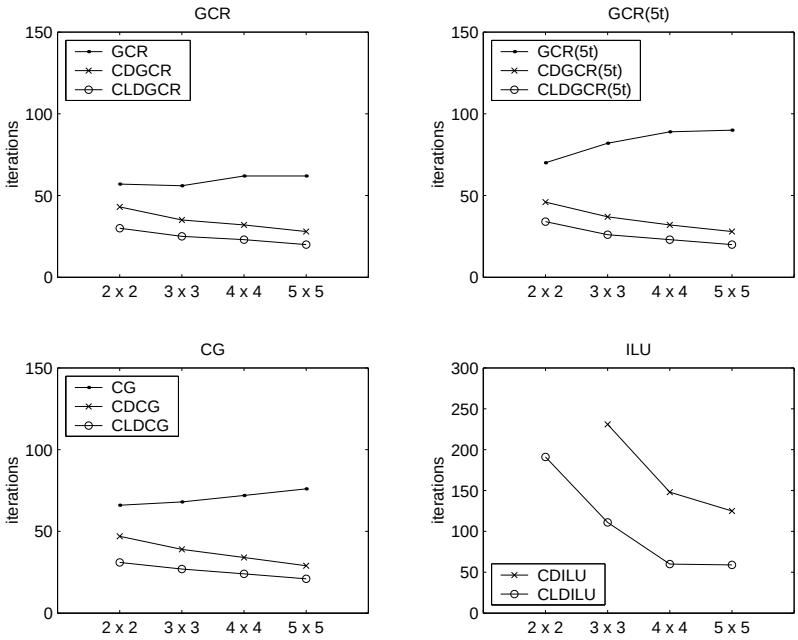

Figure 5.21: Scalability results for the solution methods, for solving the Poisson equation with homogeneous Neumann BCs on a 60×60 grid.

5.4.3 Further observations

Strange convergence behavior of the Schwarz method

For the Neumann problem, it can be observed that the additive Schwarz method behaves strangely, concerning the subdomain solution. Consider the case of 15×15 grid consisting of 3×3 subdomains. Figure 5.22 shows the convergence behavior of the additive Schwarz method, for the case the subdomain problems are solved by performing, respectively, 1, 2, 5, and 15 iterations. It can be observed that for 15 iterations, the Schwarz method does not seem to converges anymore. This is strange behavior, since one does not expect the method to converge slower for an increasing accuracy of the subdomain solutions. For comparison, Figure 5.23 shows how the Schwarz method behaves for the Dirichlet problem. No strange convergence behavior can be observed, although the convergence for 15 subdomain iterations seems slightly worse, compared to 5 iterations. However, this is likely to be caused by the fact that for each curve a different random $\boldsymbol{b}$ is taken.

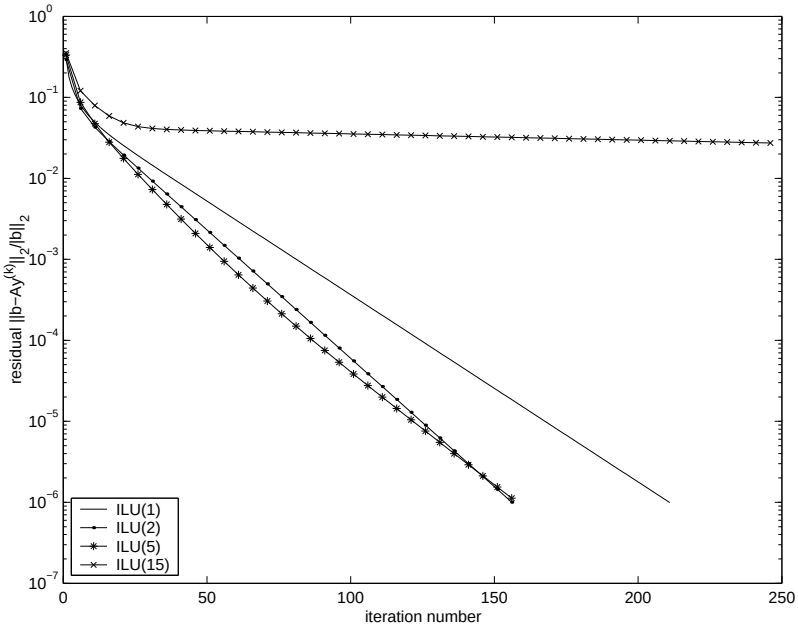

Figure 5.22: Convergence behavior of the ILU method, for an increasing number of iterations for solving the subdomain problems. The results are for the Neumann case considering a 3×3 decomposition of a 15×15 grid.

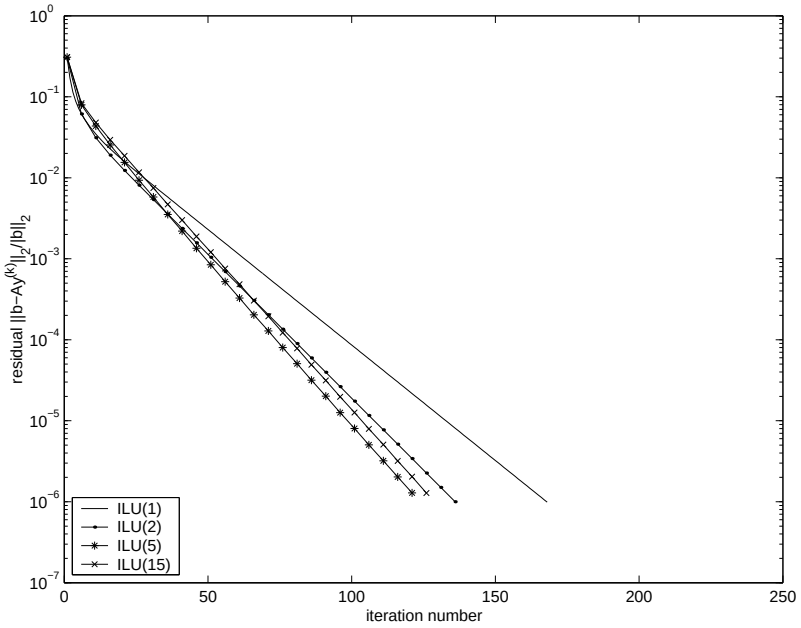

Figure 5.23: Convergence behavior of the ILU method, for an increasing number of iterations for solving the subdomain problems. The results are for the Dirichlet case considering a 3×3 decomposition of a 15×15 grid.

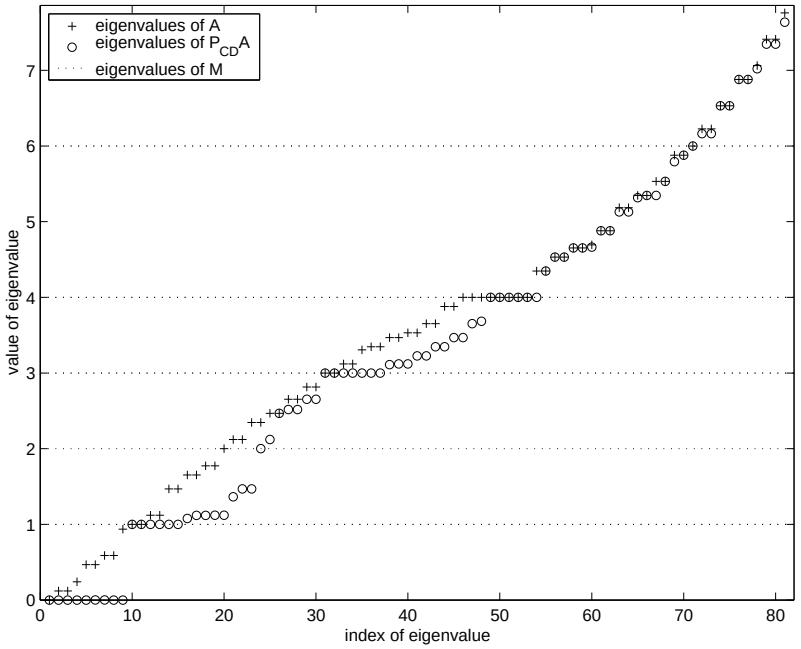

Figure 5.24: Eigenvalue spectra for the same splitting used for the Dirichlet case, but now for the Neumann case. The unpreconditioned case is considered.

A splitting of A for the 2D case and CD deflation

An interesting question is the following: does Theorem 5.1 also hold for the Neumann case, *i.e.*, for the case that A is symmetric positive semidefinite? This seems to be the case for the splitting given by (5.8).

Numerical experiments show that for this splitting, all the conditions of Theorem 5.1 are satisfied, except the condition that A should be SPD. Figure 5.24 illustrates the observation that bound (5.6) is satisfied. Figure D.5 also indicates that the preconditioned version of Theorem 5.1 holds.

A proof for these observations is left as further research. Compared to the Dirichlet case, the most crucial part in the proof appears to be the proof of $\|P\|_2 = 1$.

5.5 Connection with the X-stream code

In the X-stream code, solving the pressure-correction system (see (*iii*) in Algorithm 3.8) is most time-consuming. Therefore, our main goal is to accelerate the DD methods in X-stream, as discussed in Chapter 4. Since the pressure-correction system has an elliptic nature, we considered the Poisson equation in this chapter. Furthermore, we took an ILU additive Schwarz method, since this method is comparable to the additive Schwarz method using SIP in X-stream, which is the default solution method. Furthermore, the 2D case is considered, which gives a good indication what to expect for the 3D case.

Simple experiments with MATLAB for a pipe flow testcase (which will be described in the next chapter) show that the pressure-correction matrix is singular, having a zero eigenvalue with a corresponding eigenvector containing all ones. Experiments also indicate that the RHS vector is consistent. This is the main reason why the Neumann case is considered in this chapter. Furthermore, it is not completely clear whether the pressure-correction matrix is symmetric or not. This appears difficult to verify in a large CFD code. Therefore, we also considered the CG-Schwarz method in this chapter.

6

Deflated Krylov-Schwarz domain decomposition for the Navier-Stokes equations

6.1 Introduction

In this Chapter, Deflated Krylov-Schwarz DD is applied to the pressure-correction system in the X-stream code.

In Section 6.2, first some details are given on the implementation of both methods, followed by three testcases in Section 6.3. In Section 6.4, results for these testcases are discussed for solving the pressure-correction system. Finally, in Section 6.5 results are presented for the number of outer iterations (see Figure 3.3) and total wall-clock time.

6.2 Some implementation aspects

During the Master's project, the GCR-Schwarz and CG-Schwarz methods combined with CD and CLD deflation are implemented in the X-stream code. The implementation is done such that systems can be solved for different variables. The GCR-Schwarz method can be both truncated or restarted and MGS orthonormalization is applied. For solving the subdomain solutions, the SIP, SPTDMA and CG methods can be used. We restrict ourselves entirely to solving the subdomain problems using 1 SIP iteration.

All the implementation is done, such that the code can be run in parallel according to the MPI standard. Both Krylov-Schwarz methods without deflation run in parallel. However, the deflation routines do not work yet for the parallel case, due to an MPI routine which has to be tested first.

The implementation is done in a similar way as described in Subsection 5.3.2. We will only give a brief treatment. Consider the multi-processor case, in which each processor is assigned to exactly one subdomain. On each processor m, the matrix A_{mm} is stored as well as the coupling matrices A_{mj}, $m \neq j$, which are stored as single arrays containing only non-zero elements.

The deflation method is implemented in X-stream such that the deflation vectors are in the directions of the grid lines, instead of the Cartesian coordinates. Since the grid is structured, this can be easily implemented. Furthermore, we remove a constant deflation vector for the first subdomain to avoid singularity of E. For the CLD case, we also remove a linear deflation vector for a direction in which there is only one grid cell. This is also done to avoid singularity

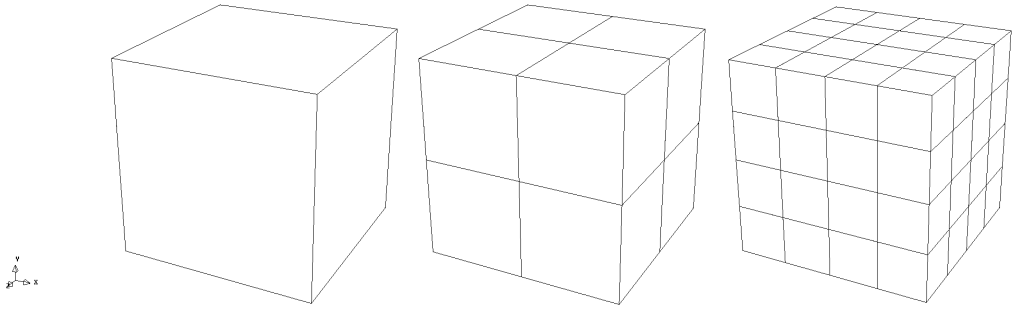

Figure 6.1: Three decomposition for the pipe flow testcase: $1 \times 1 \times 1$, $2 \times 2 \times 2$ and $4 \times 4 \times 4$.

of E.

After constructing the deflation vectors, processor m computes the corresponding block-row of AZ[1]. This requires nearest neighbor communication between processors, in which values of the deflation vectors are sent to all neighboring subdomains through the ghost cells. Next, each processor computes the non-zero row elements of E. This can be done without communication. Then, the elements of E are distributed over all processors, requiring global communication. On each processor, the matrix E is built using the non-zero pattern of E, which is stored locally. A simple LU factorization is done on each processor. In the X-stream code, both the L matrix as the U are stored, and no further attention has been given to optimize the LU factorization. Systems involving E are solved on each processor, requiring global communication for constructing the RHS vector.

6.3 Three testcases in X-stream

In Subsection 6.3.1, three testcases in X-stream are described, for which the GCR-Schwarz and CG-Schwarz methods will be extensively tested. In Subsection 6.3.2, parameter variation of the relaxation parameters is done.

6.3.1 Testcase descriptions

Varying from relatively simple to complex, the testcases we will consider are: a pipe flow, a buoyancy-driven cavity flow and a glass tank model. Only stationary, laminar flow is considered for these testcases, and all grids are orthogonal. We will restrict ourselves to the first and coarsest gridlevel for all testcases, for sake of simplicity (see Figure 3.3).

Testcase I: Pipe flow

In this testcase (X-stream reference XTC-35), the stationary incompressible Navier-Stokes equations are solved for a unit cube.

The geometry is 3D and is given by, in respectively x, y and z-direction, 1 [m] $\times$ 1 [m] $\times$ 1 [m]. The inlet is located at $x = 0$, the outlet at $x = 1$. The four remaining boundaries

[1]Note that this is slightly different from what is done in Frank & Vuik [5], where the columns of Z are stored on each processor.

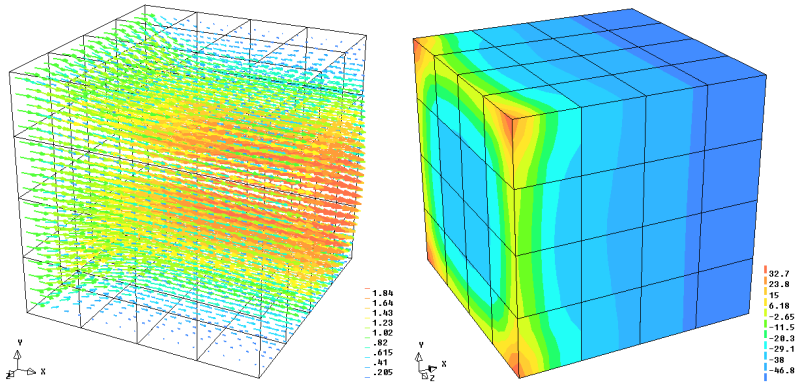

Figure 6.2: The computed velocity field (left) and pressure field (right) for the pipe flow problem.

are all walls. A uniform $16 \times 16 \times 16$ grid is taken, and we consider three decompositions: $1 \times 1 \times 1$, $2 \times 2 \times 2$ and $4 \times 4 \times 4$, as depicted in Figure 6.1.

We assume that the flow has a constant density of $\rho = 1.0$ [kg m^{-3}] and a dynamic viscosity $\mu = 1.0$ [kg m^{-1} s^{-1}]. Let u, v, and w be respectively the velocity in x, y and z direction. At the inlet, we take the Dirichlet BCs $u = 1$ [m s^{-1}], $v = w = 0$ [m s^{-1}]. At the walls, we take no-slip conditions, *i.e.*, $u = v = w = 0$ [m s^{-1}]. At the outlet, homogeneous BCs are taken for u, v and w. By this, the Reynolds' number is 1, which makes the flow laminar. Furthermore, we assume no gravitation contribution and an initial velocity and pressure field of zero. Figure 6.2 shows the resulting velocity and pressure field, respectively.

Testcase II: Buoyancy-driven cavity flow

The equations to be solved for this testcase (X-stream reference XTC-33) are the stationary incompressible Navier-Stokes equations and the stationary energy equation.

The geometry is a 1 [m] $\times$ 1 [m] 2D plate. Since the X-stream code is 3D instead of 2D, we take one grid cell in the z-direction and apply a symmetry condition. By this, we assume that we solve a 2D problem in X-stream. We will take a 60×60 uniform grid and the following three decompositions: 1×1, 2×2 and 4×4, see Figure 6.3.

The basic idea of a buoyancy-driven cavity flow is, that free convection occurs due to

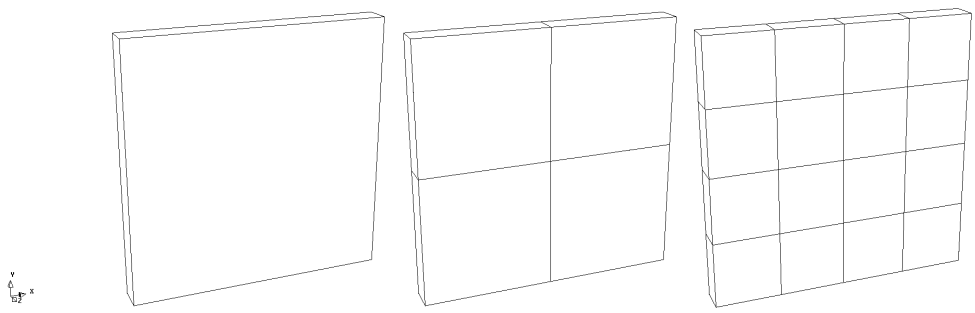

Figure 6.3: Three decompositions for the buoyancy-driven cavity flow: 1×1, 2×2 and 4×4.

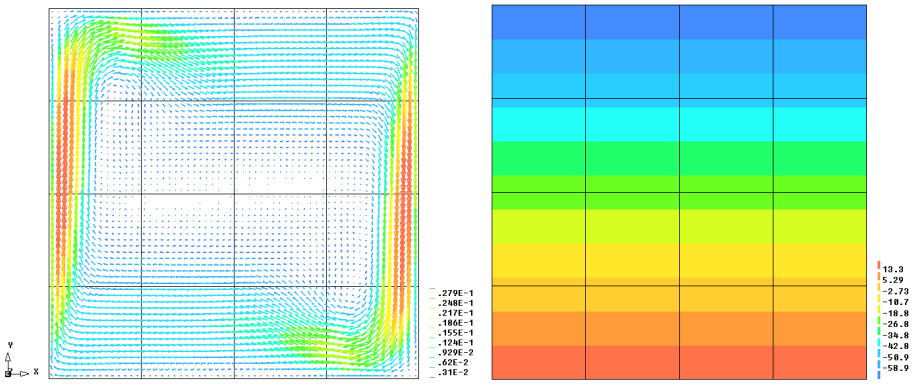

Figure 6.4: The velocity field (left) and pressure field (right) for the buoyancy-driven cavity flow.

a temperature difference between a 'hot' wall and a 'cold' wall. The heated fluid is rising along the hot wall, while cooled fluid is falling along the cold wall. Let T_1 be denoted as the temperature of the left hot wall, and T_0 the temperature of the right cold wall. We assume that the Boussinesq approximation holds, and that the density varies linear with temperature:

$$\rho(T) = \rho_0 - \rho_0\beta(T_1 - T_0) \ ,$$

where ρ_0 is the reference density. The rate of circulation of the flow depends on the Rayleigh number, which is defined for a unit square as

$$\mathrm{Ra} = \frac{\rho^2 g\beta(T_1 - T_0)}{\mu^2}\mathrm{Pr} \ ,$$

with Pr denotes the Prandtl number. We take the constants

$$\begin{aligned} \mu &= 1.0 \cdot 10^{-3} \ [\mathrm{kg\ m^{-1}\ s^{-1}}] \ , \\ c_p &= 7.1 \cdot 10^{2} \ \ [\mathrm{m^2\ s^{-2}\ K^{-1}}] \ , \\ \lambda &= 1.0 \qquad\ \ [\mathrm{kg\ m\ s^{-3}\ K^{-1}}] \ , \end{aligned}$$

where c_p is the specific heat and λ the heat conductivity, and by this we have Pr $= 0.71$. Furthermore, we choose

$$\begin{aligned} g &= 9.81 \ \ [\mathrm{m\ s^{-2}}] \ , \\ \beta &= 0.001 \ [\mathrm{K^{-1}}] \ , \end{aligned}$$

where g is the gravitation constant. Taking $T_1 = 1.436$ [K] and $T_0 = 0$ [K] for the temperature BCs, we have Ra $= 1.0 \cdot 10^6$. We also assume that the top and bottom walls are adiabatic, and therefore take homogeneous Neumann BCs for the temperature. The flow is said to be laminar. For the velocities we take $u = v = 0$ at the wall, *i.e.*, a no-slip condition. Figure 6.4 shows the resulting velocity and pressure fields.

Testcase III: Glass tank model

For this testcase (X-stream reference XTC-07), the stationary incompressible Navier-Stokes equations and the stationary energy equation are solved. The flow considered is a buoyant laminar flow. The density, as well as the dynamic viscosity, depends on the temperature.

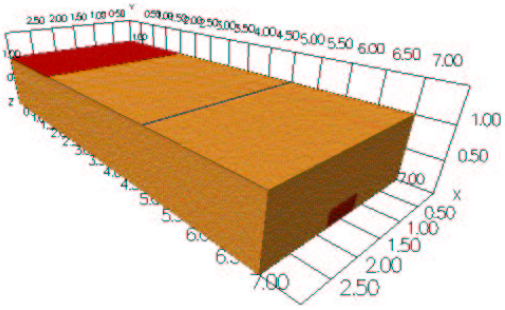

Figure 6.5: Geometry of the glass tank.

The geometry of the glass tank is as in Figure 6.5. The overall dimensions in respectively x, y and z-direction is 7 [m] $\times$ 1 [m] $\times$ 3 [m]. The red areas correspond to the inlet of the glass fractions and the outlet of the melted glass. The glass inlet runs from $x = 0$ [m] to $x = 1$ [m] over the entire width of the tank. The glass outlet is located at $x = 7$ [m] and is centered in z-direction with a width of 0.6 [m] and height of 0.25 [m]. A grid with a total of 10,500 grid cells is used, and the domain is decomposed into 18 blocks, as depicted in Figure 6.6.

Since the testcase is complex, we restrict ourselves only to a global treatment of the BCs. At the inlet, homogeneous Dirichlet BCs are taken for the velocities and a linear profile Dirichlet condition for the temperature in the x-direction. At the glass outlet, a homogeneous Neumann condition is taken for all velocities. At the glass surface, homogeneous Dirichlet conditions are taken for the velocity components and a piecewise linear profile Dirichlet condition in the x-direction. For the walls, no-slip conditions for the velocities are taken and a homogeneous Neumann condition for the temperature.

Figure 6.7 shows the velocity and pressure field. Note that the velocities point upward at the glass surface. This is physically incorrect, because no glass could leave the tank at the surface. Probably, this is caused by the insufficiency of the PWI method for buoyant flows, or by the fact that linear extrapolation of the pressure at the glass surface is not accurate enough.

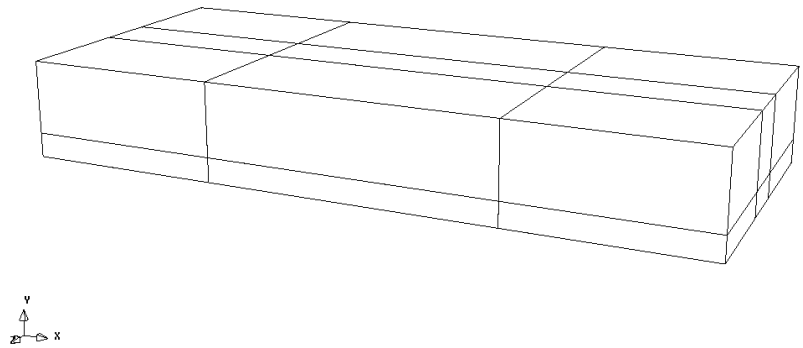

Figure 6.6: Subdomain topology for the glass tank model, in which the domain is decomposed into 18 blocks.

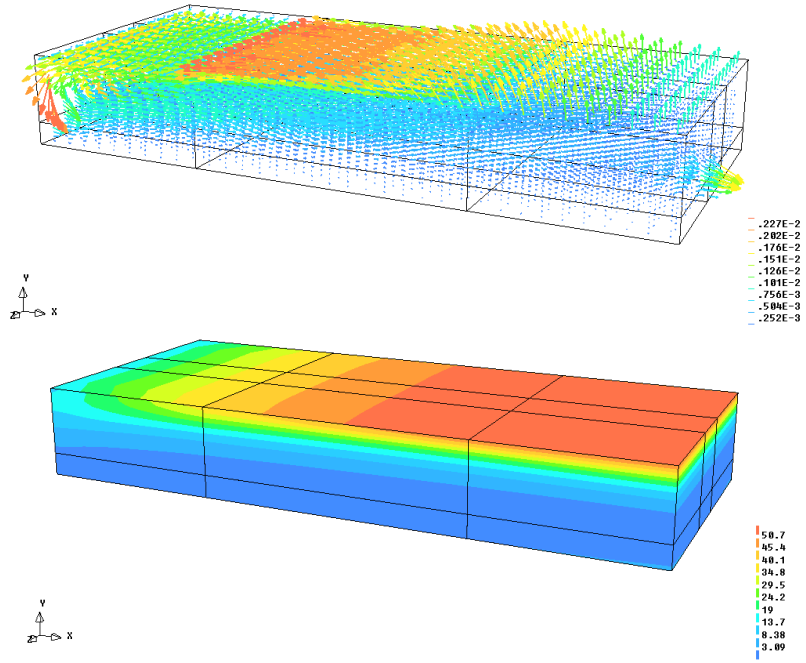

Figure 6.7: Velocities (top) and pressure (bottom) for the glass tank model.

6.3.2 Parameter variation

When solving a complex problem, usually several variables need to be underrelaxed (see Section 3.3). How the relaxation parameters have to be chosen can be a difficult and time-consuming task, especially when a large number of variables are being solved and the grid is large. This is because the settings are directly linked to the convergence behavior of the outer iteration process. When one chooses the parameters too high, the risk exists that the method diverges. When one chooses them too low, this can result in a large number of outer iterations, and therefore, into large wall-clock times. This is the reason why parameter variation is of importance in order to obtain a convergent result, and in a minimal number of iterations and computing time. Parameter variation has been done for the three testcases.

For the parameter variation we take, more or less, the default settings of X-stream. For solving all the variables, the additive Schwarz method is used with inaccurate subdomain solution obtained by one SIP step. From now on, we will refer to this method as simply SIP(1). All variables, except the pressure-correction, are solved by 5 inner iterations. The pressure-correction system is solved by 10 iterations of SIP(1). All the inner iterations are performed, without applying a termination criterion. Convergence is reached if the residuals are smaller than 10^{-6}. No effort has been taken to look at the monitor points for convergence. The number of SSI iterations for Testcase I has been taken equal to 1, for Testcase II and II this value is 10.

Testcase I: Pipe flow

Parameter variation is done for a $4 \times 4 \times 4$ decomposition. For this testcase, we have under-relaxation parameters for the pseudo-velocities and the pressure. For simplicity, we consider the relaxation parameter for the three pseudo-velocities to be the same, and denote this by α_u. The relaxation parameter for the pressure is called α_p. The maximum number of outer iterations is set to 500. Figure 6.8 shows the outer iterations as well as the wall-clock times for the relaxation parameter variation. Optimal values of α_u and α_p appear to be given by the following table:

α_u	α_p
0.7	0.5

Table 6.1: Relaxation parameters for Testcase I obtained by parameter variation, considering a $4 \times 4 \times 4$ decomposition.

Testcase II: Buoyancy-driven cavity flow

Parameter variation is done for a 4×4 decomposition. The involving relaxation parameters are: α_u, α_p, α_ρ and α_e, where α_ρ respectively α_e are the relaxation parameters for the density and energy.

The maximum number of iterations was set to 5000. Figure 6.9 shows the outer iterations for different combinations of these parameters. A value at 5000 iterations means that the convergence criterion was not satisfied after 5000 iterations. It can be observed that the number of outer iterations is proportional to the wall-clock time, see Figure E.1 of Appendix

E. This is what to be expected, since all outer iterations are equally expensive, due to the fact that the termination criterion for the inner iterations is never used. Optimal values for α_u, α_p, α_ρ and α_e appear to be:

α_u	α_p	α_ρ	α_e
0.5	0.7	1.0	1.0

Table 6.2: Relaxation parameters for Testcase II obtained by parameter variation, considering a 4×4 decomposition.

The fact that the density does not need to be underrelaxed, is probably because the variation in the density is small due to the large reference density ρ_0.

Testcase III: Glass tank model

The involving relaxation parameters are: α_u, α_p, α_ρ, α_e and α_μ, where α_μ is the relaxation parameter for the dynamic viscosity. For this testcase, the maximum number of outer iterations was set to 1000. Figure 6.10 shows the result of the parameter variation analysis. Figure E.2 in Appendix E shows again that the computing time is proportional to the outer iterations. The following relaxation parameters seem to be optimal:

α_u	α_p	α_ρ	α_e	α_μ
0.7	0.3	0.1	1.0	0.1

Table 6.3: Relaxation parameters for Testcase III obtained by parameter variation.

It can be observed that the convergence of the outer iterations is very sensitive to the choice of α_e. It was observed that parameter variation for $\alpha_e = 0.9$ resulted in outer iteration residuals which stayed constant.

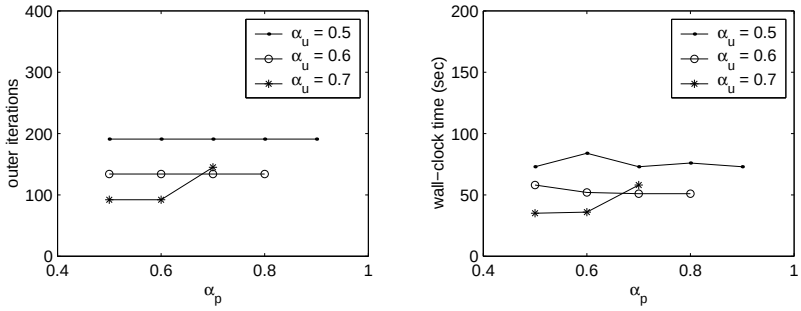

Figure 6.8: Results for the parameter variation of the relaxation parameters. Left: concerning outer iterations; right: concerning total wall-clock times. Testcase I is considered for a $4 \times 4 \times 4$ decomposition.

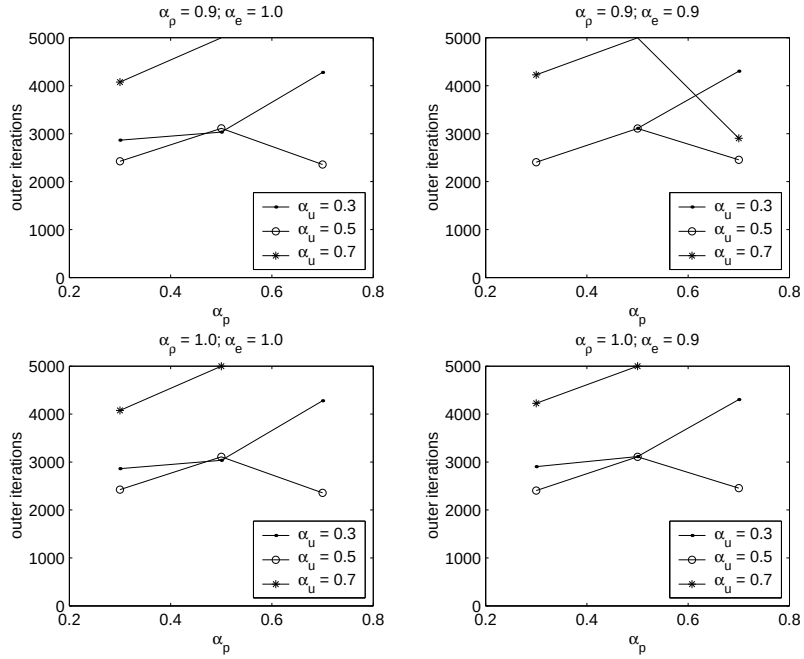

Figure 6.9: Parameter variation for the number of outer iterations. Testcase II is considered for a 4×4 decomposition.

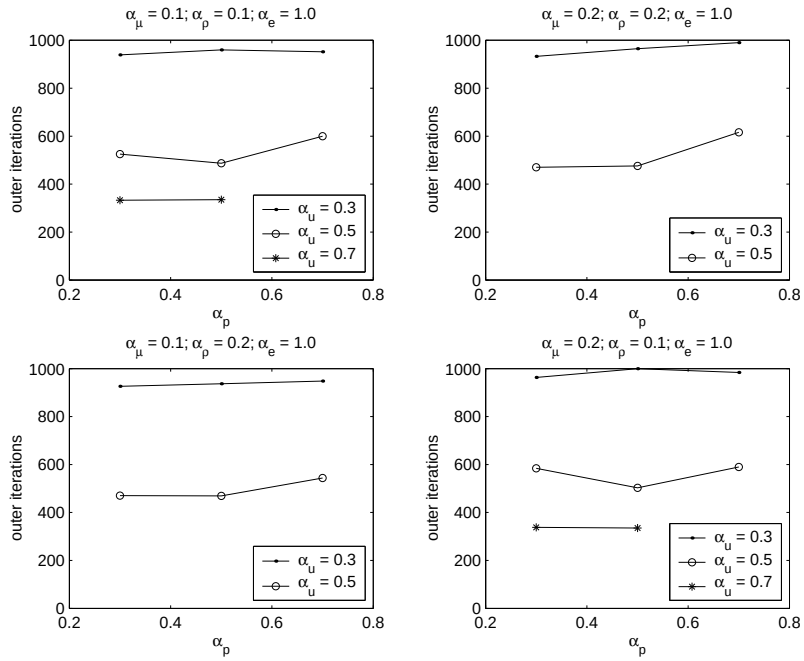

Figure 6.10: Parameter variation for the number of outer iterations, considering Testcase III.

6.4 Results for the number of inner iterations

In this section, the GCR-Schwarz method and the CG-Schwarz method, with and without CD and CLD deflation, are compared to three iterative solution methods used in X-stream: the SIP(1) method, the SPTDMA(1) method and the CG(1) method. The SIP(1) method is default in X-stream and most commonly used. The CG(1) method is not often used, since this method can only be used for SPD matrices.

First, the solution of the Poisson equation in X-stream is discussed briefly in Subsection 6.4.1. Then, the results for the three testcases are presented in Subsection 6.4.2 for solving the pressure-correction system. All the timing results in this chapter are done on a Pentium III machine.

6.4.1 Solving the 3D Poisson equation

Consider the 3D Poisson equation with homogeneous Dirichlet BCs for the grid of Testcase I with a $4 \times 4 \times 4$ decomposition, see Subsection 6.3.1. The pressure-correction matrix for this testcase was used to construct the Poisson matrix in a 'quick and dirty' way, by simply overwriting the main diagonal elements with 6 and the non-zero off-diagonal elements with -1. The RHS vector was build by using a simple combination of the grid cell indices.

Figure 6.11 shows the results for the three current methods in X-stream, for the case of 30 inner iterations. Figure 6.12 respectively Figure 6.13, show the results for the GCR-Schwarz and CG-Schwarz, using no deflation, CD deflation and CLD deflation. The results are as to be expected from the numerical experiments done in Chapter 5.

We see a significantly large improvement in convergence rate compared to the SIP(1), SPTDMA(1) and CG(1) additive Schwarz methods. Furthermore, it can be observed that the GCR-Schwarz and CG-Schwarz methods with deflation have a significantly better performance than without deflation. However, this seems likely to be caused by the large number of subdomains, *i.e.*, 64 in total. Note the small jump of the residual in the first iteration for the CLD deflation case.

Table E.1 in Appendix E.2 gives a complete comparison for all the methods concerning inner iterations and wall-clock times. The left table shows the termination criteria for all the solution methods after 30 iterations. The right table shows the required iterations and wall-clock time for a solution method to obtain the same termination criterion as SIP(1) has after 10 and 20 iterations respectively. Also the GCR-Schwarz method with restarting and truncation is considered. It turns out that truncation for only one search direction is sufficient, see Figure 6.14.

When we take a closer look at the timing results in Table E.1 (right table), we see that the CG-Schwarz and GCR-Schwarz methods with CD deflation are more efficient than without deflation. The methods with CD deflation are also quite competitive compared to the solution methods in X-stream. Furthermore, we conclude from this table that CLD deflation is very expensive for this case, up to a factor 5 more wall-clock time compared to the case without deflation. Although this table gives us a good indication, it should be noted that the measured times are very small and likely sensitive to disturbances.

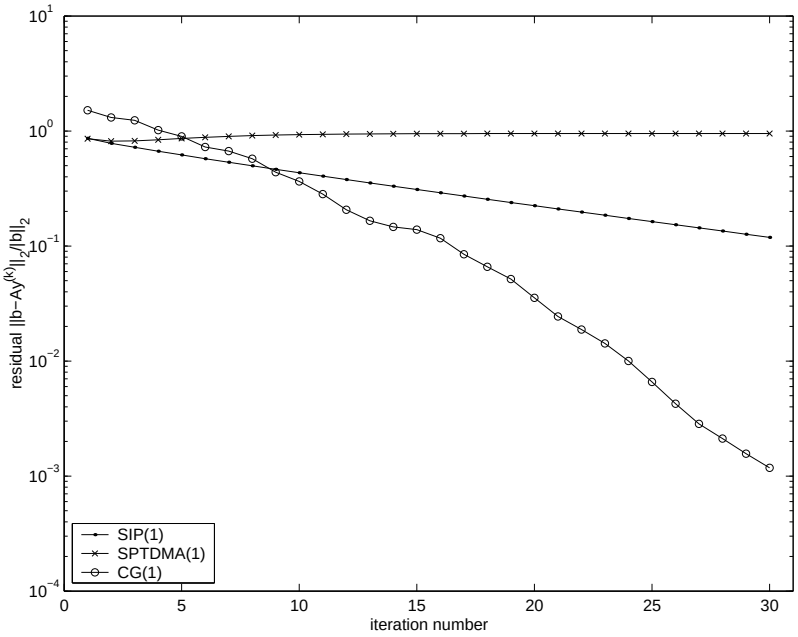

Figure 6.11: The residuals of the SIP(1), SPTDMA(1) and CG(1) methods for solving the Poisson equation on the same grid as for Testcase I, using a $4 \times 4 \times 4$ decomposition.

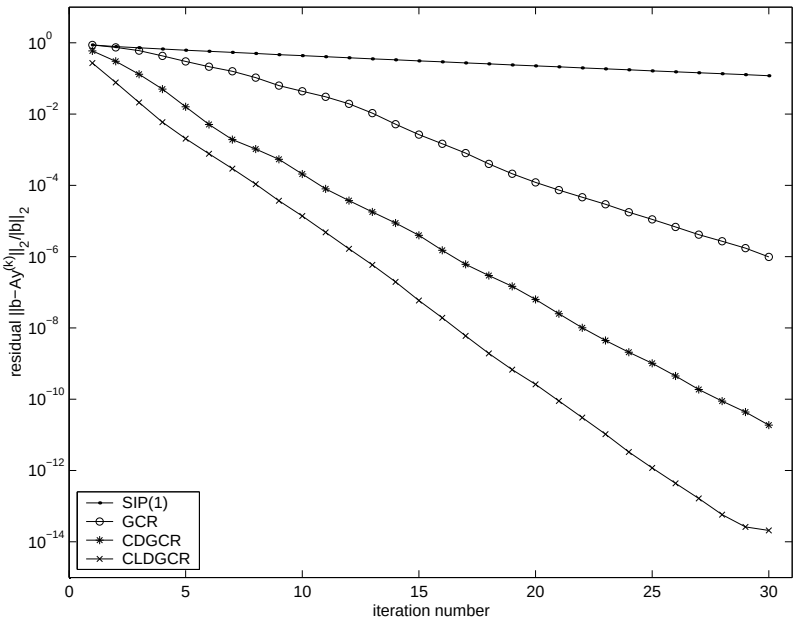

Figure 6.12: The residuals of the SIP(1), GCR, CDGCR and CLDGCR methods for solving the Poisson equation on the same grid as for Testcase I, using a $4 \times 4 \times 4$ decomposition.

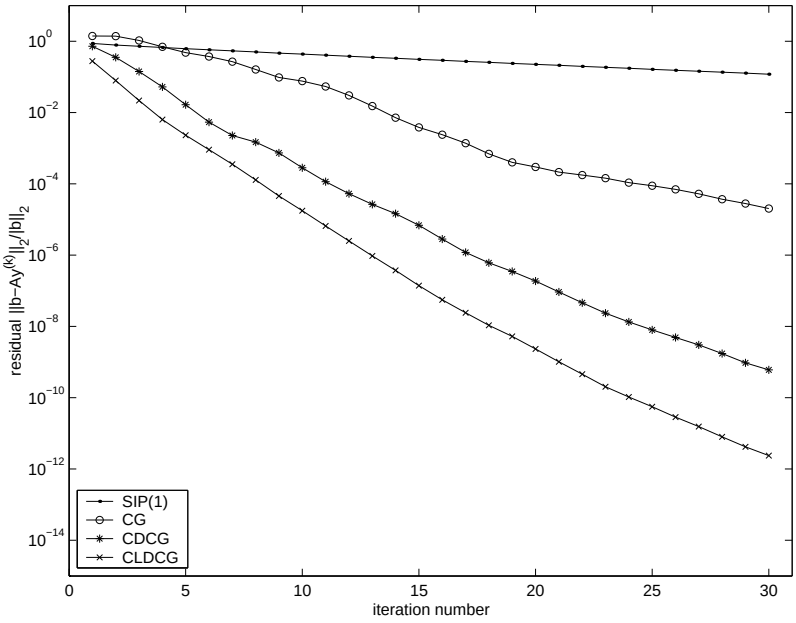

Figure 6.13: The residuals of the SIP(1), CG, CDCG and CLDCG methods for solving the Poisson equation on the same grid as for Testcase I, using a $4 \times 4 \times 4$ decomposition.

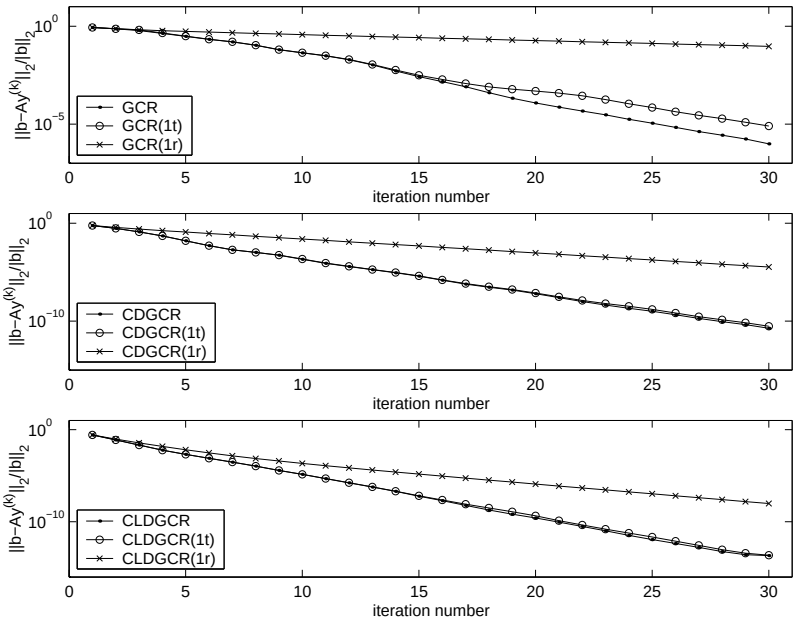

Figure 6.14: GCR-Schwarz methods compared for restarting and truncating for one search direction. Top plot: GCR; middle plot: CDGCR; bottom plot: CLDGCR.

6.4.2 Solving the pressure-correction system

For all the results presented in this subsection the determined relaxation parameters as in Subsection 6.3.2 are used.

Testcase I: Pipe flow

Consider the case of a $4 \times 4 \times 4$ decomposition. The results presented here are for the pressure-correction system in the second outer iteration. This is done, since it can be observed that the convergence behavior of all the solution methods for the first outer iteration is significantly different from the other iterations. This is likely caused of the zero initial fields.

Figure 6.15 shows the results for the three solution methods in X-stream, for the case of 30 inner iterations. Figures 6.16 and 6.17, respectively, show the results for the GCR-Schwarz and CG-Schwarz methods, using no deflation, CD deflation and CLD deflation. The results are very similar to the results for solving the Poisson equation. Again, we see good convergence behavior with CD and CLD deflation compared to the current solution methods in X-stream. We also see that, in general, the GCR method performs better than the CG method.

Tables E.2 and E.3 in Appendix E.2 show more details on the comparisons for the three decompositions. Also truncated and restarted GCR-Schwarz is considered. Just as for the Poisson problem, it can be observed that truncation for only one search direction is most efficient, see Figure E.3 of Appendix E. From Table E.3, we conclude that the CD and CLD method make the convergence behavior more or less independent on the number of subdomains. This is illustrated by Figure 6.18. Note that the GCR-Schwarz and the CG-Schwarz method without deflation do not have good scalability properties. Furthermore, note that both Krylov-Schwarz methods have good performances for the case of 1 subdomain.

Concerning wall-clock times, we conclude from Table E.3 that the CLD deflation becomes more expensive for an increasing number of subdomains. For 1 subdomain, deflation is more expensive than no deflation, and truncated GCR-Schwarz seems to be most efficient. For 64 subdomains, CLD becomes very expensive, and truncated GCR appears to be the most efficient method considering 20 iterations of SIP(1).

In Figure 6.16 it can been seen that the convergence for the CLDGCR method stagnates from iterate 23. Possibly, this strange effect is caused by removing one constant deflation vector from Z. Figure 6.20 shows the convergence behavior for the case we do not remove a constant deflation vector, but change one element into $1 + \varepsilon$, where $\varepsilon = 0.001$. For the GCR-Schwarz method, we observe that the residuals become constant at a certain iteration, after which it declines again. The effect in Figure 6.16 seems quite similar, although less extreme. Possibly, this effect depends on the number of grid cells per subdomain (see Section 5.4).

Analogous to our observations in Chapter 5, it can be observed that it is advisable to take the zero starting vector for solving the subdomain problems. Figure 6.19 shows the result for the GCR-Schwarz method when we take an old, orthonormalized, search direction as the starting vector. Comparing this figure to Figure 6.16, we observe that this choice has a negative influence on the performance of the deflation methods. For the CG-Schwarz method, we see that the residuals even stay horizontal, see Figure E.6 in Appendix E.

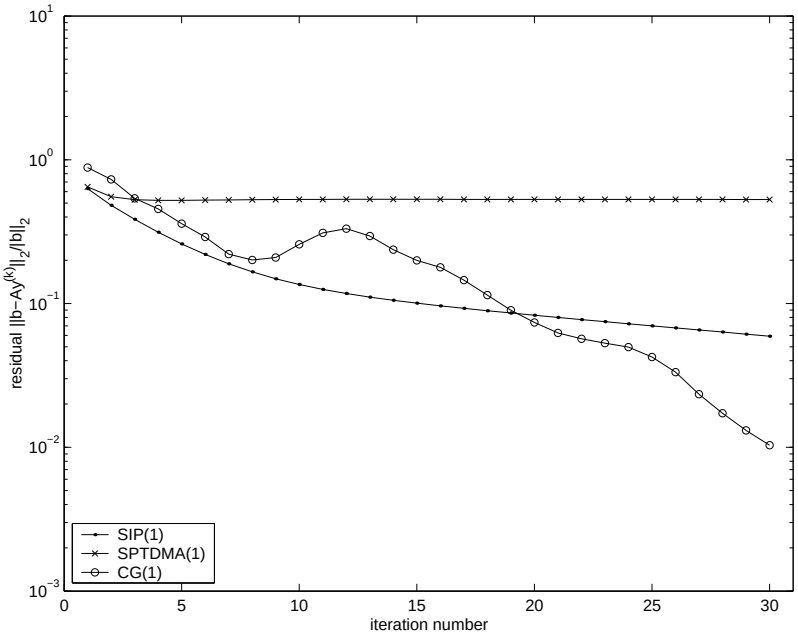

Figure 6.15: The residuals for the SIP(1), SPTDMA(1) and CG(1) methods for solving the pressure-correction system. Testcase I is considered for a $4 \times 4 \times 4$ decomposition.

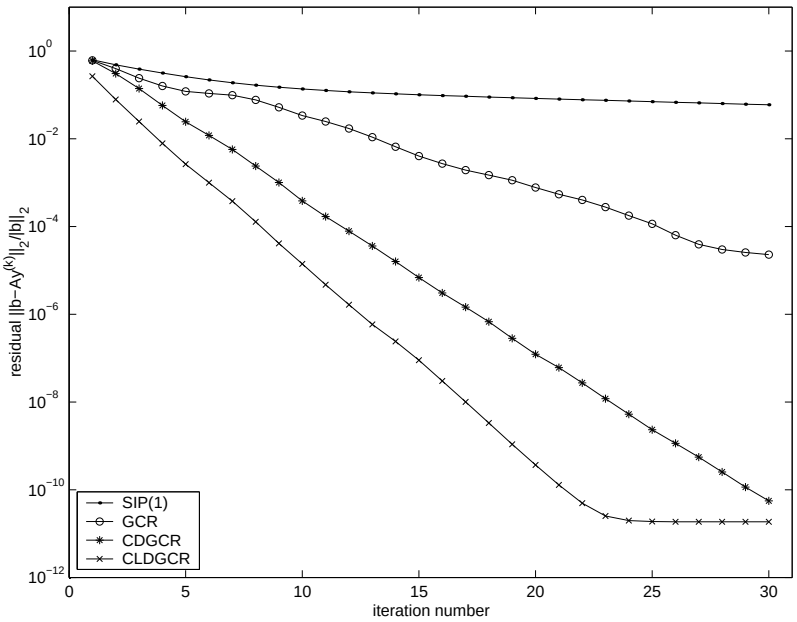

Figure 6.16: The residuals for the SIP(1), GCR, CDGCR and CLDGCR methods for solving the pressure-correction system. Testcase I is considered for a $4 \times 4 \times 4$ decomposition.

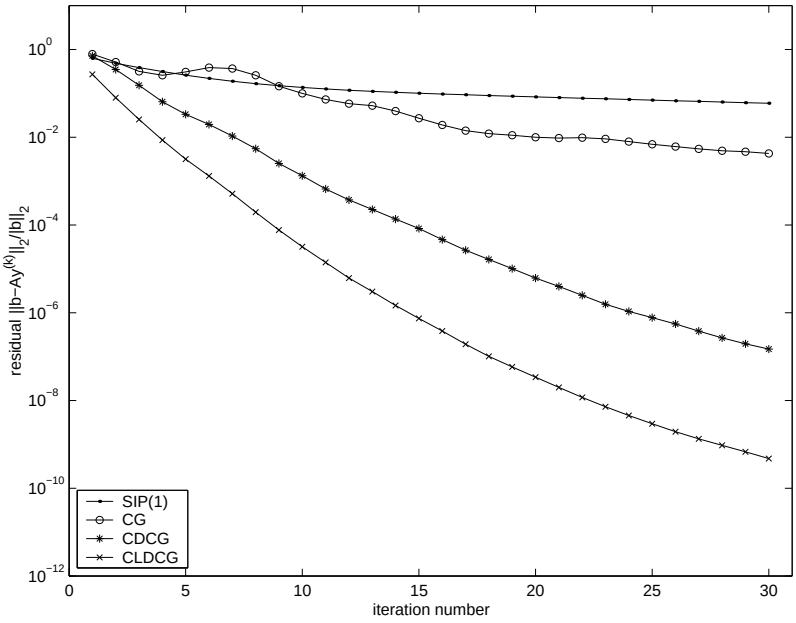

Figure 6.17: The residuals for the SIP(1), CG, CDCG and CLDCG methods for solving the pressure-correction system. Testcase I is considered for a $4 \times 4 \times 4$ decomposition.

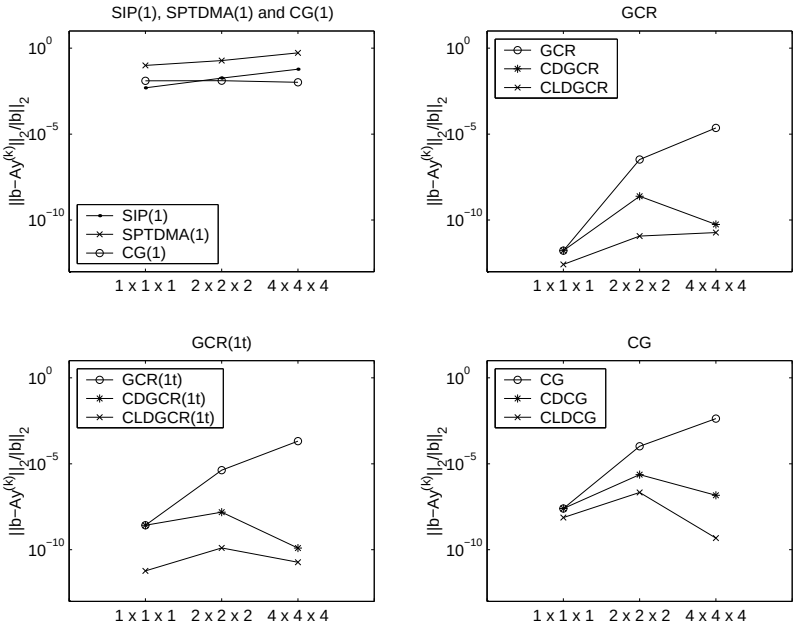

Figure 6.18: Scalability results for Testcase I considering three decompositions. The pressure-correction system is considered.

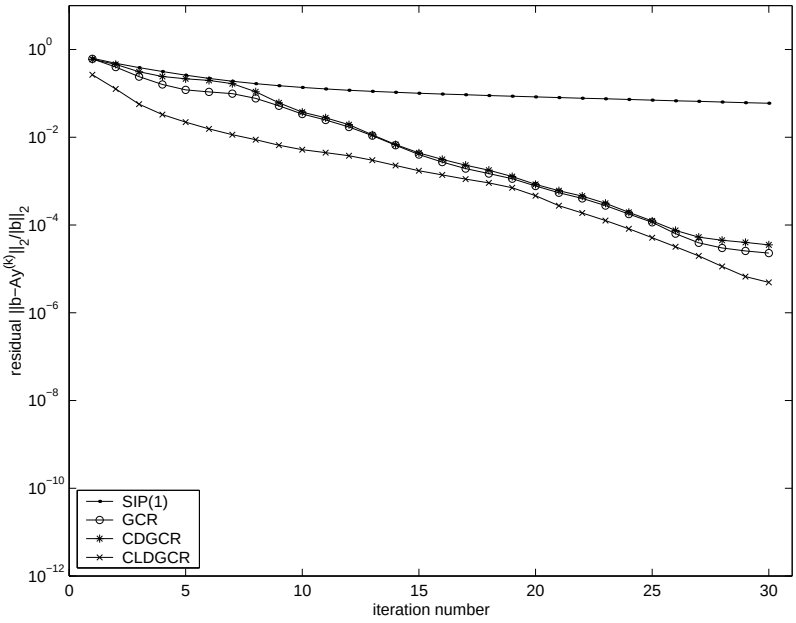

Figure 6.19: Convergence behavior for the SIP(1), GCR, CDGCR and CLDGCR methods, when an old search direction is taken as a starting vector. Testcase I is considered for a $4 \times 4 \times 4$ decomposition.

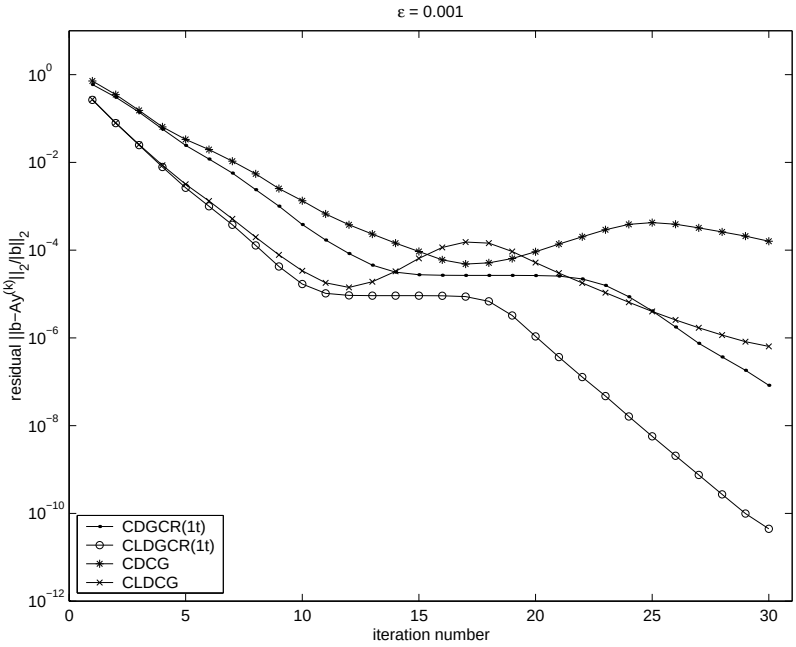

Figure 6.20: The residuals for the GCR(1t)-Schwarz and CG-Schwarz methods when one element in the constant deflation vector is set to $1 + 10^{-3}$. The pressure-correction system for Testcase I is considered using a $4 \times 4 \times 4$ decomposition.

Testcase II: Buoyancy-driven cavity flow

Consider the case of a 4×4 decomposition, then Figures 6.21, 6.22 and 6.23 show the results for solving the pressure-correction system using, respectively, the current solution methods in X-stream, the GCR-Schwarz method and the CG-Schwarz method.

From Figure 6.21, we conclude that the X-stream solution methods converge very poorly for this testcase. Figure 6.22 shows a good converge behavior for the GCR-Schwarz method with CD and CLD deflation. Note the jump of the residual for CLD deflation in the first iteration. Figure 6.23 shows that the CG-Schwarz method performs less well than the GCR-Schwarz method. This observation, together with the fact that the CG(1) has a bad convergence behavior, seems to indicate that the pressure-correction matrix is possibly not symmetric for this testcase.

Tables E.4 and E.5 in Appendix E.2 show a comparision of the solution methods for three different decompositions. Figure 6.24 shows the scalability results. We clearly see the effect that the convergence rate improves for an increasing number of subdomains, when considering the Deflated GCR-Schwarz method. This is especially the case for CLD deflation. Again, we see from Table E.5 that only one truncation vector is more or less sufficient for the GCR-Schwarz method, especially when deflation is applied, see Figure E.4 of Appendix E. Concerning computing time, we conclude from Table E.5 that restarted and truncated GCR-Schwarz with CLD deflation is the most efficient method combination for all decompositions.

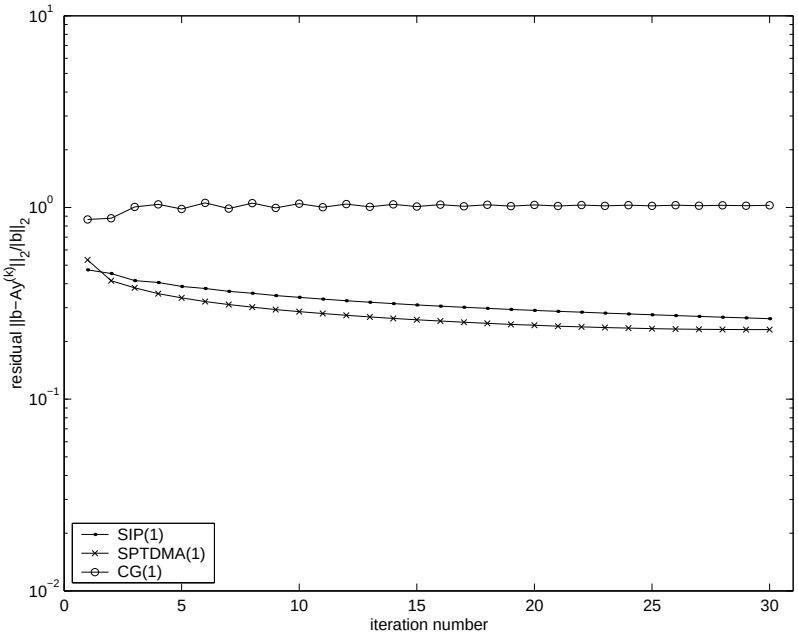

Figure 6.21: The residuals of the SIP(1), SPTDMA(1) and CG(1) methods for solving the pressure-correction system. Testcase II is considered for a 4×4 decomposition.

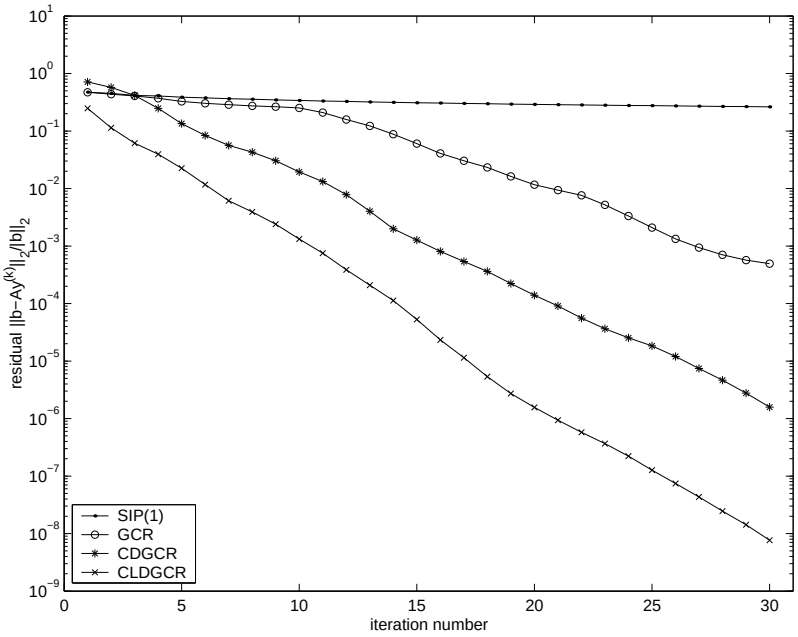

Figure 6.22: The residuals of the SIP(1), GCR, CDGCR and CLDGCR methods for solving the pressure-correction system. Testcase II is considered for a 4×4 decomposition.

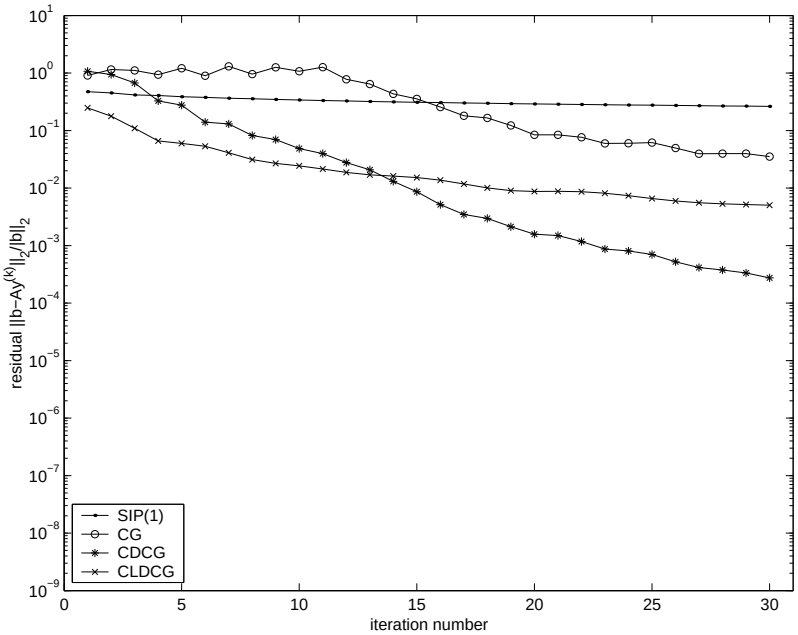

Figure 6.23: The residuals of the SIP(1), CG, CDCG and CLDCG methods for solving the pressure-correction system. Testcase II is considered for a 4×4 decomposition.

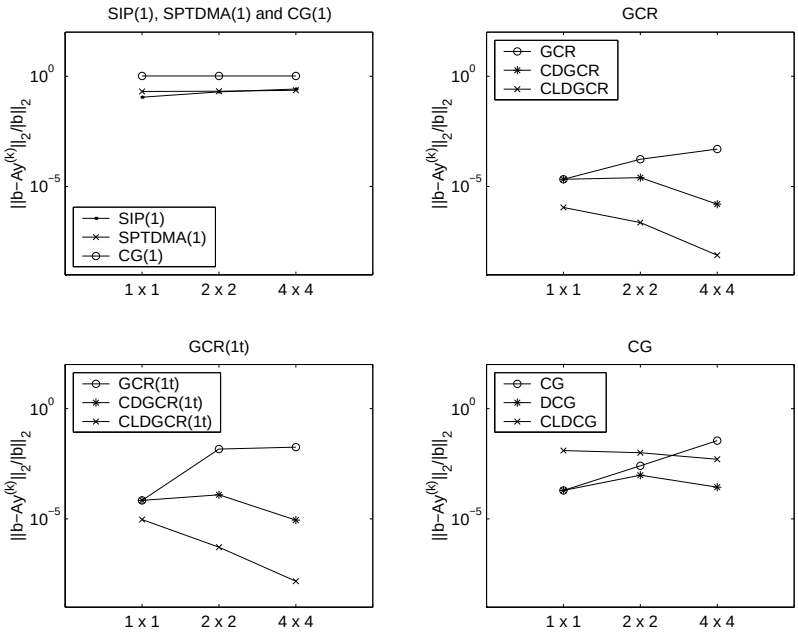

Figure 6.24: Scalability results for Testcase II considering three decompositions. The pressure-correction system is considered.

Testcase III: Glass tank model

Figures 6.25, 6.26 and 6.27 show the results for solving the pressure-correction system using, respectively, the solution methods of X-stream, the GCR-Schwarz and the CG-Schwarz methods.

In Figure 6.25, we see that the SIP(1) method converges quite well, compared to Testcase I and II. In Figure 6.26, it can be observed that the GCR-Schwarz with CD deflation has a similar convergence rate compared to the case of no deflation. Furthermore, a relatively large jump in the residual in the first iteration for the CLDGCR-Schwarz method can be observed. A similar jump can be seen in Figure 6.27 for the CLDCG-Schwarz method.

Table E.6 in Appendix E.2 shows a comparison for all the solution methods. Concerning iterations, truncated GCR-Schwarz for one search direction is most efficient, see Figure E.5 in Appendix E. Concerning wall-clock time, this method is most efficient compared to 20 iterations with the SIP(1) method.

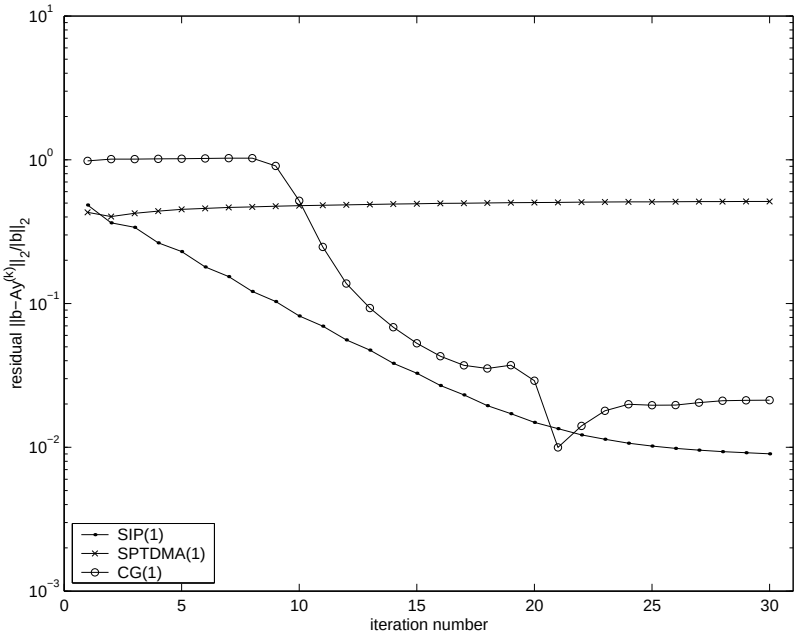

Figure 6.25: The residuals of the SIP(1), SPTDMA(1), and CG(1) methods for solving the pressure-correction system. Testcase III is considered.

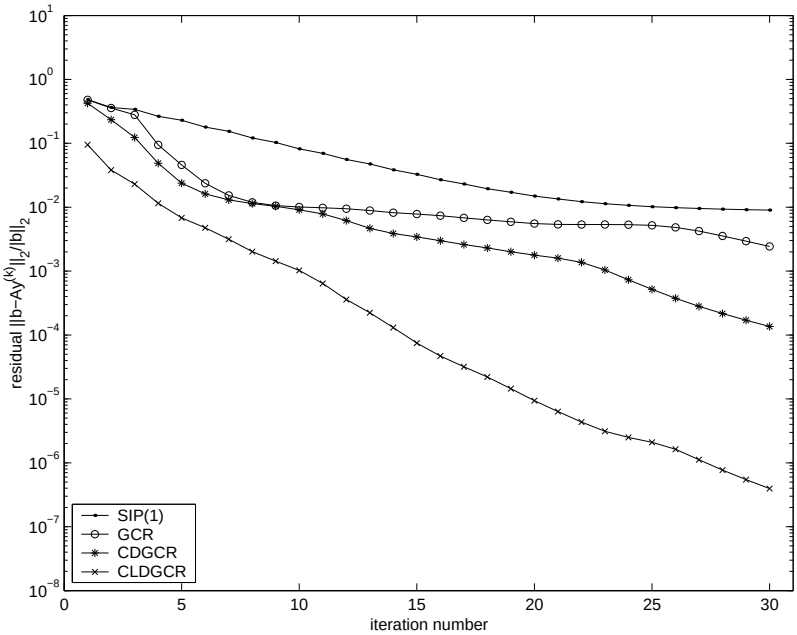

Figure 6.26: The residuals of the SIP(1), GCR, CDGCR, and CLDGCR methods for solving the pressure-correction system. Testcase III is considered.

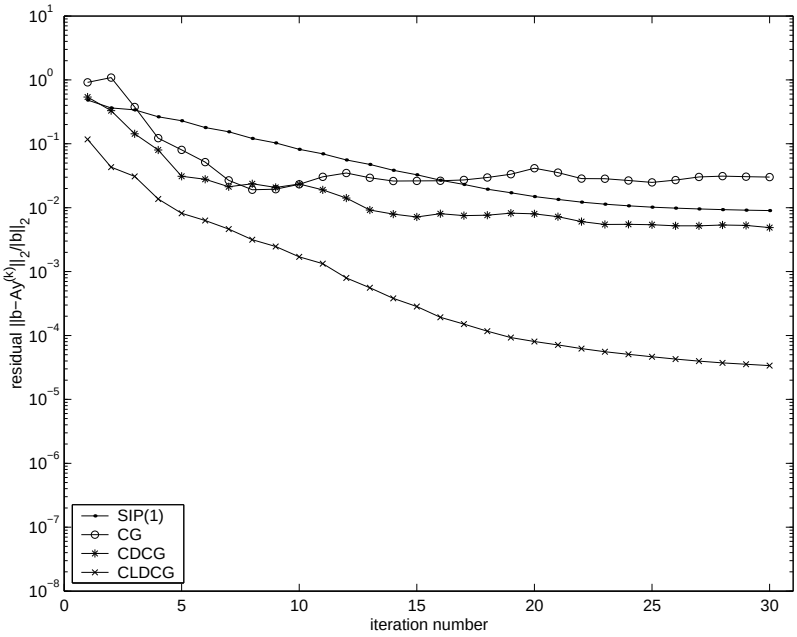

Figure 6.27: The residuals of the SIP(1), CG, CDCG and CLDCG methods for solving the pressure-correction system. Testcase III is considered.

6.5 Results for the number of outer iterations

In Subsection 6.4.2, results were presented for the number of inner iteration for solving the pressure-correction system. An interesting question is the following: how is the solution of the pressure-correction system for different solution methods related to the total number of outer iterations and total wall-clock time?

Our approach to answer this question is as follows. We take the underrelaxation parameters for the three testcases as in Subsection 6.3.2. We will vary the number of inner iterations for the pressure-correction using the SIP(1) method and the GCR-Schwarz method for the case of no deflation, CD deflation and CLD deflation. Furthermore, we will vary the number of SSI iterations. The effect for different settings on the total number of outer iterations and the total wall-clock time is investigated. We will restrict ourselves to the treatment of the GCR-Schwarz method for which truncation is done for one search direction, see Subsection 6.4.2. All timing results presented in this chapter were obtained on a Pentium IV 2.53 GHz machine with 1 Gb of internal memory.

Testcase I: Pipe flow

We restrict ourselves to the case of a $4 \times 4 \times 4$ decomposition. Experiments show that for the other decompositions the number of outer iterations is similar using SIP(1) and the same relaxation parameters. However, the wall-clock time is largest for the case of 64 subdomains.

Figure 6.30 shows the results for the number of outer iterations. For the case of deflation, a significant insensitivity can be observed. For the SIP(1) and the GCR(1t) methods without deflation this does not hold. In Figure 6.31, the corresponding wall-clock times are depicted. We see that the GCR(1t)-Schwarz method with CLD is most expensive, followed by CD deflation and the case of no deflation. In Figure 6.28, the outer iteration residuals are given for the case of 10 inner iterations SIP(1), and SSI = 1. In Figure 6.29, the same is done for the CDGCR(1t) method, taking 1 inner iteration and SSI = 1. For these settings the SIP(1) method and the CDGCR(1t) method appear optimal. Note that less oscillations in the residuals for the case of CD deflation can be observed.

Figure 6.32 shows the percentages time for solving the pressure correction system, of the total wall-clock time, using the SIP(1) method and GCR(1t)-Schwarz method. It can be observed that the percentage time increases for an increasing number of inner iterations. In Figure 6.33, the same is done for the CDGCR(1t)-Schwarz and CLDGCR(1t)-Schwarz methods (top plots). We see that the CLDGCR-Schwarz method takes a large part of the total solution time.

Interesting is to know the amount of work for solving the system due to the deflation methods. This is also illustrated in Figure 6.33 (bottom plots). We observe that CLD deflation is more expensive than CD deflation. Furthermore, a decrease in percentages for an increasing number of inner iterations can be observed. One possible explanation for this behavior is that for less inner iterations, the initialization of the deflation method becomes expensive.

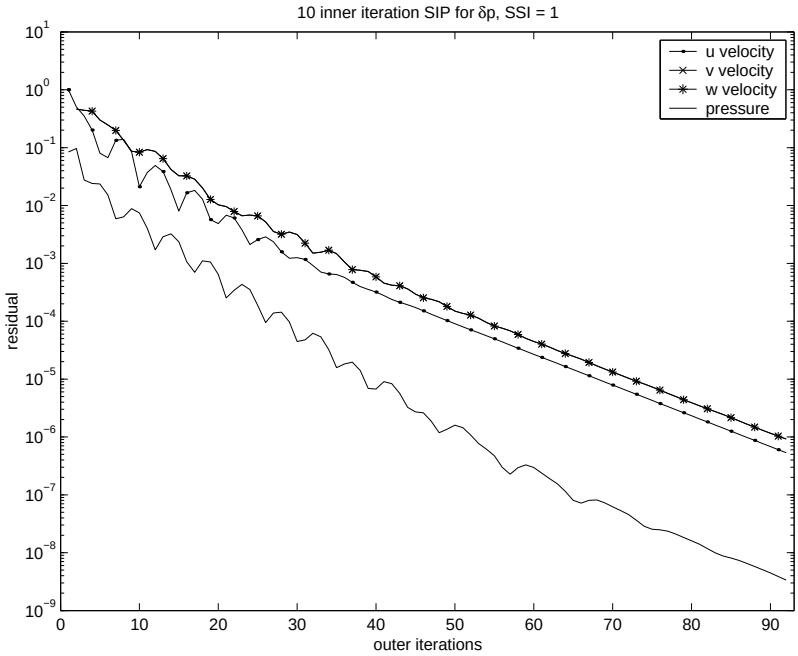

Figure 6.28: The residuals of the outer iterations, taking 10 inner iterations with the SIP(1) method for solving the pressure-correction system, and SSI = 1. Testcase I is considered for a $4 \times 4 \times 4$ decomposition.

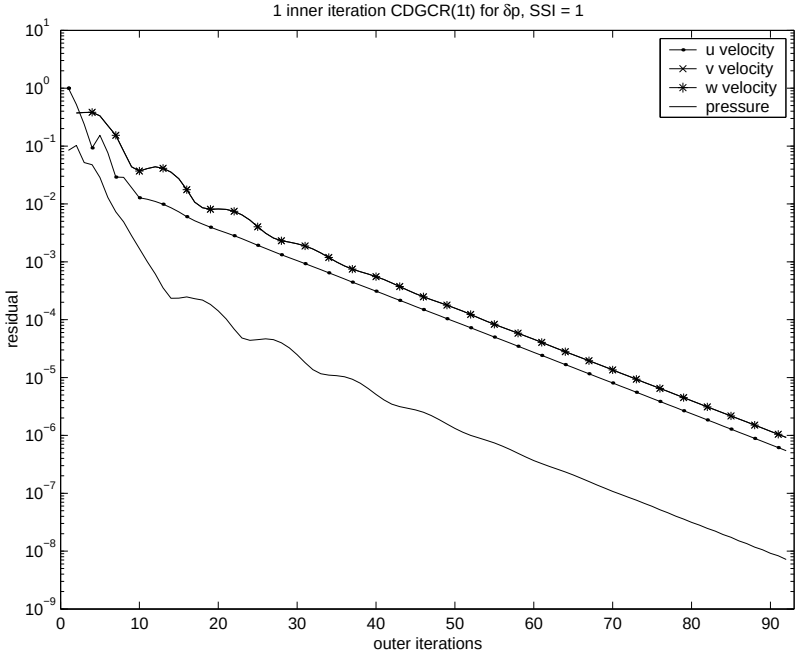

Figure 6.29: The residuals of the outer iterations, taking 1 inner iteration with the CDGCR(1t) method for solving the pressure-correction system, and SSI = 1. Testcase I is considered for a $4 \times 4 \times 4$ decomposition.

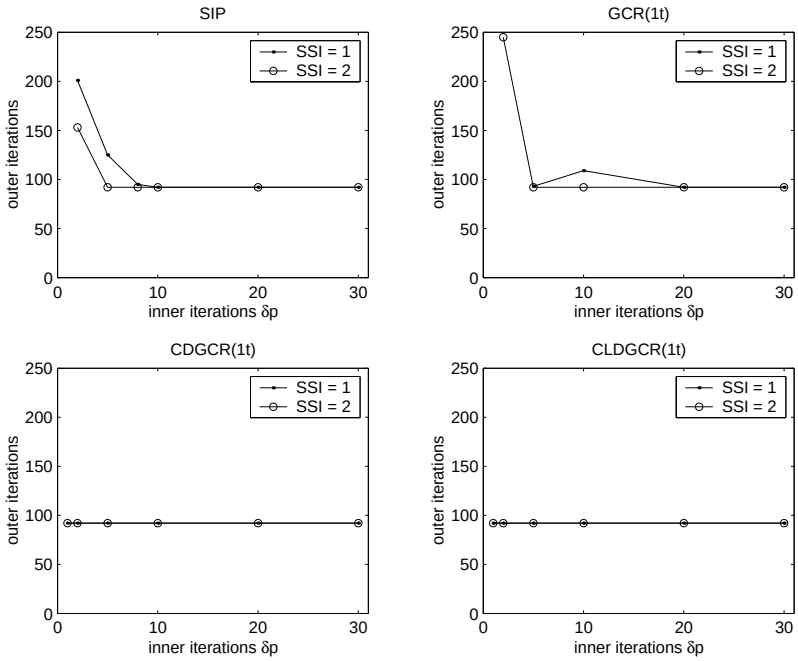

Figure 6.30: The number of outer iterations for different values of pressure-correction inner iterations and SSI iterations, comparing the SIP(1), GCR(1t), CDGCR(1t) and CLDGCR(1t) methods. Testcase I is considered for a $4 \times 4 \times 4$ decomposition.

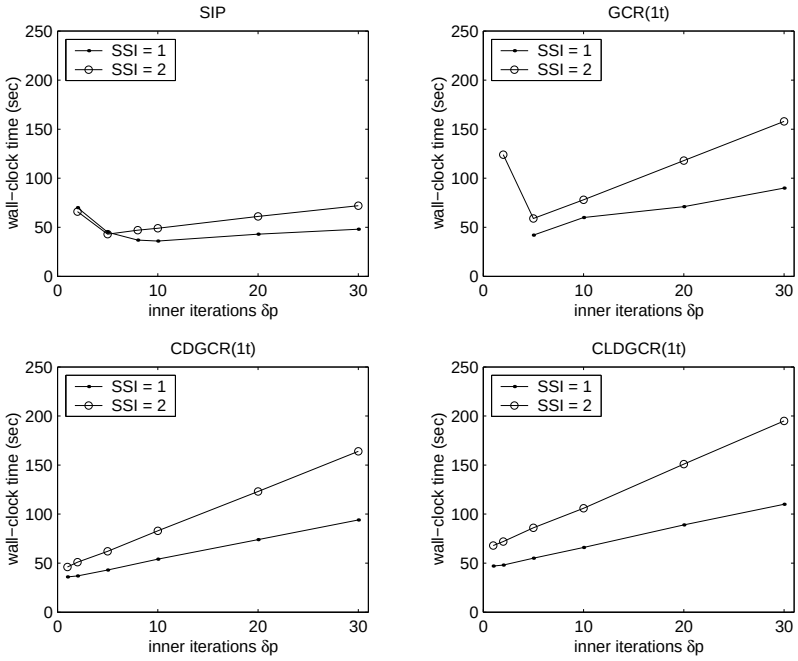

Figure 6.31: The total wall-clock time for different values of pressure-correction inner iterations and SSI iterations, comparing the SIP(1), GCR(1t), CDGCR(1t) and CLDGCR(1t) methods. Testcase I is considered for a $4 \times 4 \times 4$ decomposition.

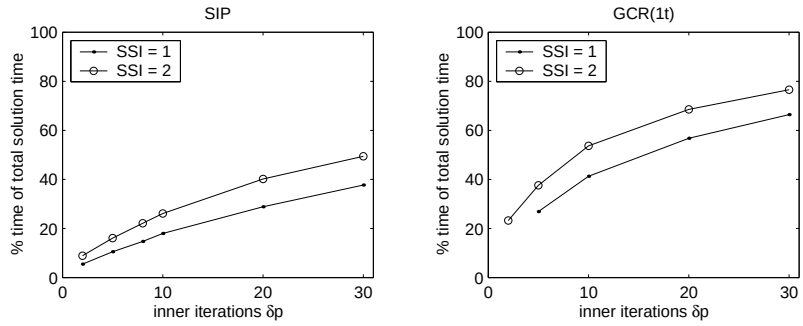

Figure 6.32: The percentage time for solving the pressure-correction system of the total wall-clock time, for the SIP(1) and GCR(1t) methods. Testcase I is considered for a $4 \times 4 \times 4$ decomposition.

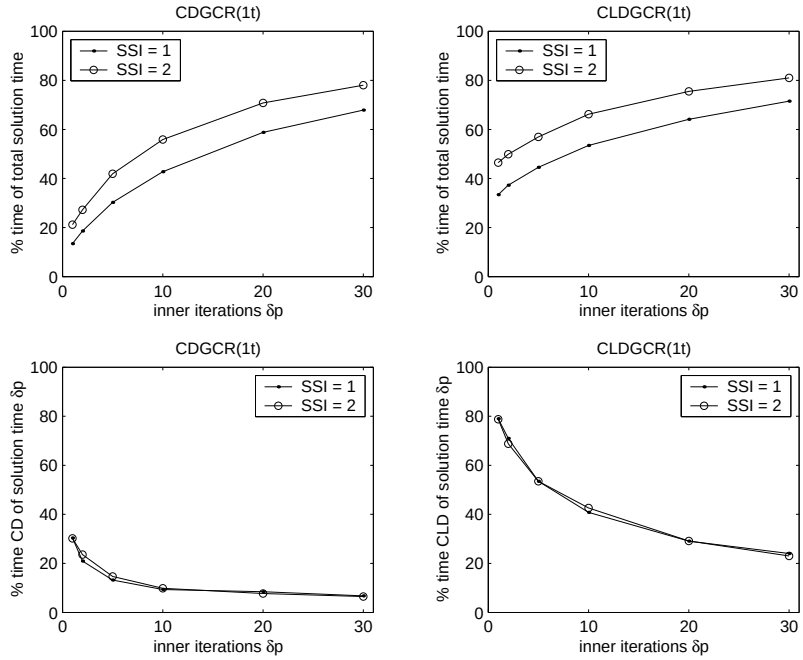

Figure 6.33: Top: the percentage time for solving the pressure-correction system of the total wall-clock time, for the CDGCR(1t) and CLDGCR(1t) methods. Bottom: the percentage time for deflation of the time for solving the pressure-correction system, for the CDGCR(1t) and CLDGCR(1t) methods. Testcase I is considered for a $4 \times 4 \times 4$ decomposition.

Testcase II: Buoyancy-driven cavity flow

We restrict ourselves to a 4×4 decomposition. It can be noted that large differences in outer iterations and wall-clock times can be observed for the three different decompositions.

In Figure 6.36, the results for the number of outer iterations are given. It can be observed that the outer iterations are constant for the CLD case for different settings of the number of pressure-correction iterations and SSI iterations. Hence, we observe a similar behavior as for Testcase I. The same behavior can, more or less, be seen with CD deflation. For GCR(1t) and SIP(1) without deflation this can not be observed. In Figure 6.37, the corresponding wall-clock times are depicted. We conclude that the wall-clock times for CD and CLD deflation are practically the same. In Figure 6.34, the outer iteration residuals are given for the case of 10 inner iterations SIP(1), and SSI = 9. These are the optimal values for the SIP method. In Figure 6.35, the same is done for the optimal values of the CLDGCR(1t) method, *i.e.*, taking 1 inner iteration and SSI = 1. We observe a much better convergence behavior with less oscillations, and a gain in wall-clock time of a factor 4.

Figure 6.38 gives the percentages time for solving the pressure correction system, of the total wall-clock time, using the SIP(1) method and GCR(1t)-Schwarz method. In Figure 6.39, the same is done for the CDGCR(1t)-Schwarz and CLDGCR(1t)-Schwarz methods and the percentages due to deflation are given. We conclude that the percentages of the total wall-clock times for these methods do not differ much. The bottom plots of Figure 6.39 show that CLD is approximately twice as expensive as CD deflation. However, this effect seems negligible.

Testcase III: Glass tank model

Figure 6.42 shows the results for the number of outer iterations. Again, we see that the outer iterations are insensitive to the setting of the CLDGCR(1t) iterations for solving the pressure correction and the SSI number. In Figure 6.43, the corresponding wall-clock times are depicted, showing that CD and CLD deflation are practically equally expensive. In Figure 6.40, the outer iteration residuals are given for the case of 20 inner iteration SIP(1), and SSI = 1. Figure 6.41 shows the residuals for one CLDGCR(1t)-Schwarz iteration and SSI = 1. We observe that the convergence behavior is slightly better compared to the SIP(1) method, since a lower outer iteration termination criterion can be reached in less iterations. However, more oscillations in the residuals for the energy can be observed.

Figure 6.44 shows the percentages time for solving the pressure correction system of the total wall-clock time, using the SIP(1) method and GCR(1t)-Schwarz method. In Figure 6.33, the same is done for the CDGCR(1t)-Schwarz and CLDGCR(1t)-Schwarz methods and the percentages due to deflation are given. Again, we see that the GCR-Schwarz methods are equally expensive.

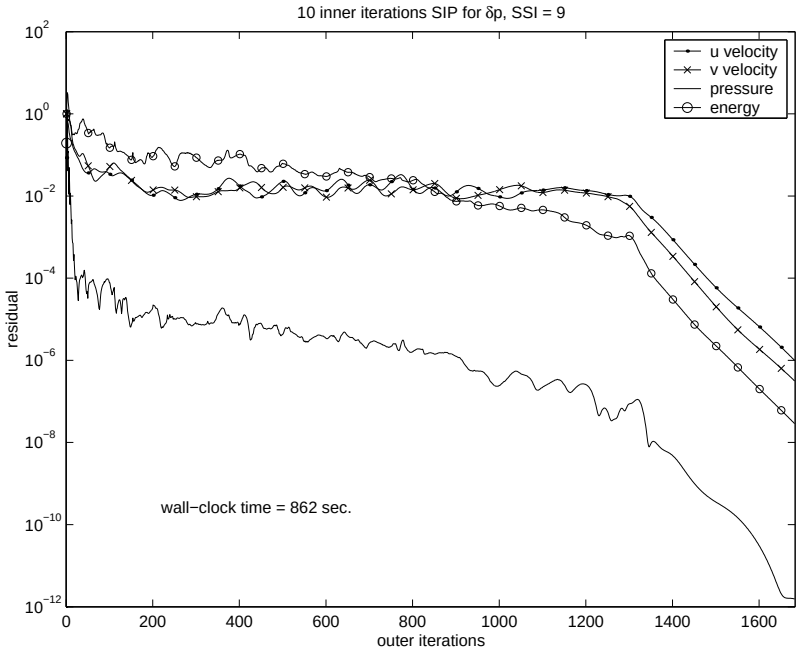

Figure 6.34: The residuals of the outer iterations, taking 10 inner iterations with the SIP(1) method for solving the pressure-correction system, and SSI = 9. Testcase II is considered for a 4×4 decomposition.

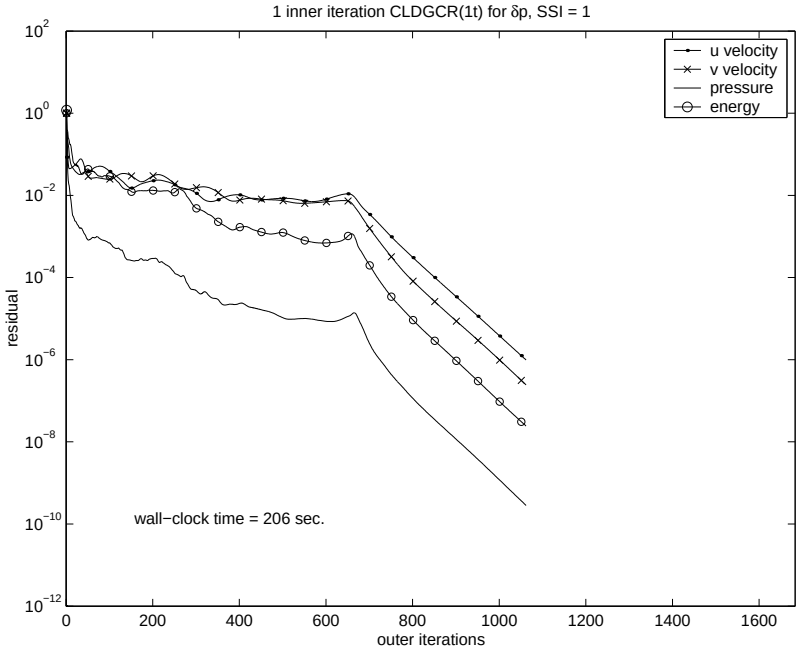

Figure 6.35: The residuals of the outer iterations, taking 1 inner iteration with the CLDGCR(1t) method for solving the pressure-correction system, and SSI = 1. Testcase II is considered for a 4×4 decomposition.

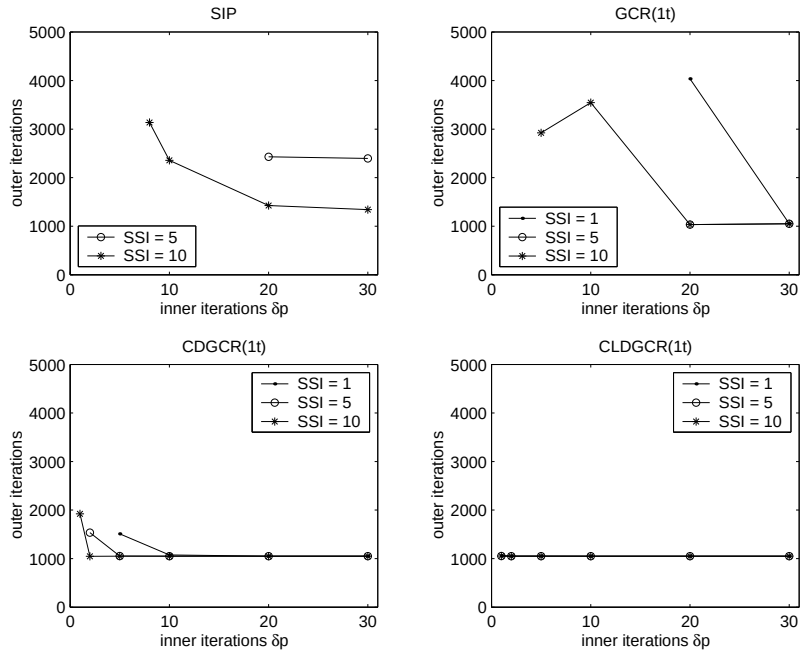

Figure 6.36: The number of outer iterations for different values of pressure-correction inner iterations and SSI iterations, comparing the SIP(1), GCR(1t), CDGCR(1t) and CLDGCR(1t) methods. Testcase II is considered for a 4×4 decomposition.

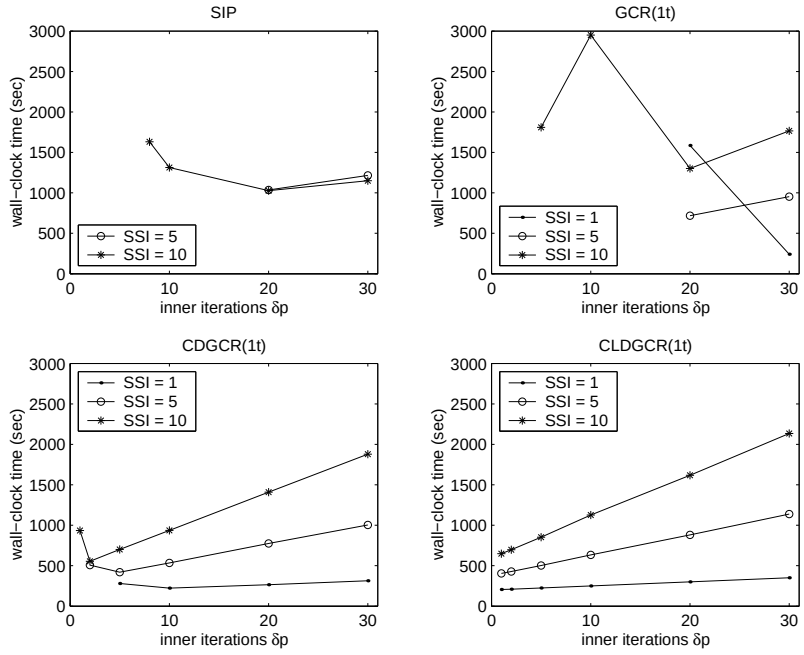

Figure 6.37: The total wall-clock time for different values of pressure-correction inner iterations and SSI iterations, comparing the SIP(1), GCR(1t), CDGCR(1t) and CLDGCR(1t) methods. Testcase II is considered for a 4×4 decomposition.

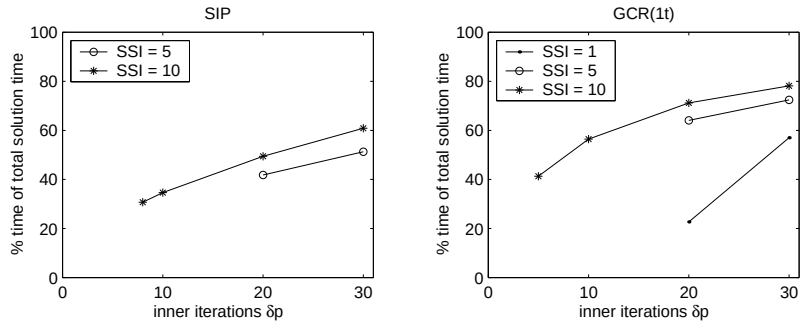

Figure 6.38: The percentage time for solving the pressure-correction system of the total wall-clock time, for the SIP(1) and the GCR(1t) method. Testcase II is considered for a 4×4 decomposition.

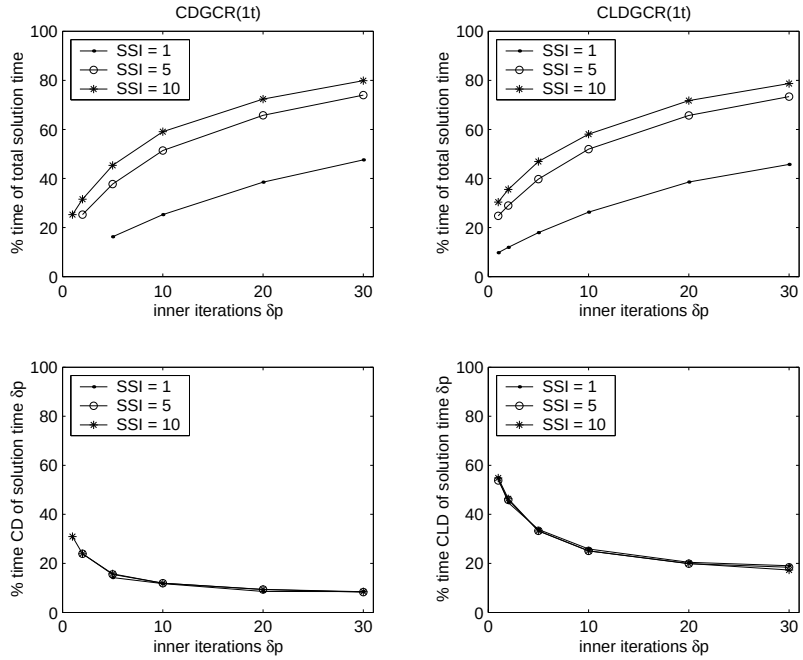

Figure 6.39: Top: the percentage time for solving the pressure-correction system of the total wall-clock time, for the CDGCR(1t) and CLDGCR(1t) methods. Bottom: the percentage time for deflation of the time for solving the pressure-correction system, for the CDGCR(1t) and CLDGCR(1t) methods. Testcase II is considered for a 4×4 decomposition.

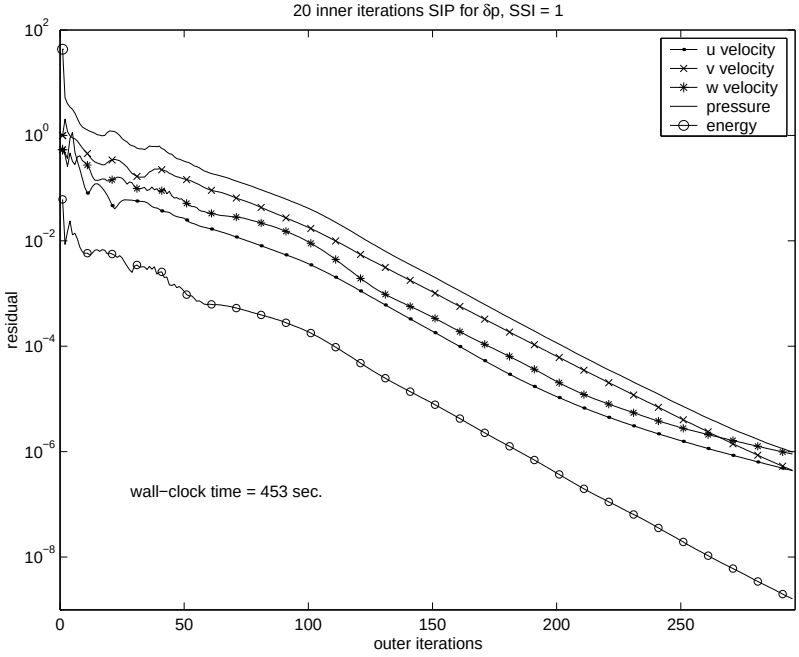

Figure 6.40: The residuals for the outer iterations, taking 10 inner iterations with the SIP(1) method for solving the pressure-correction system, and SSI = 1. Testcase III is considered.

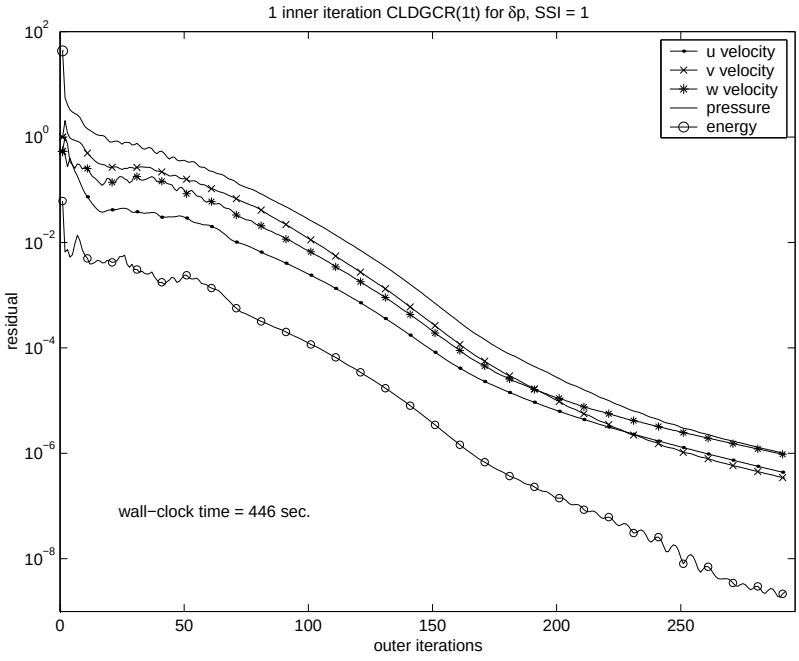

Figure 6.41: The residuals of the outer iterations, taking 1 inner iteration with the CLDGCR(1t) method for solving the pressure-correction system, and SSI = 1. Testcase III is considered.

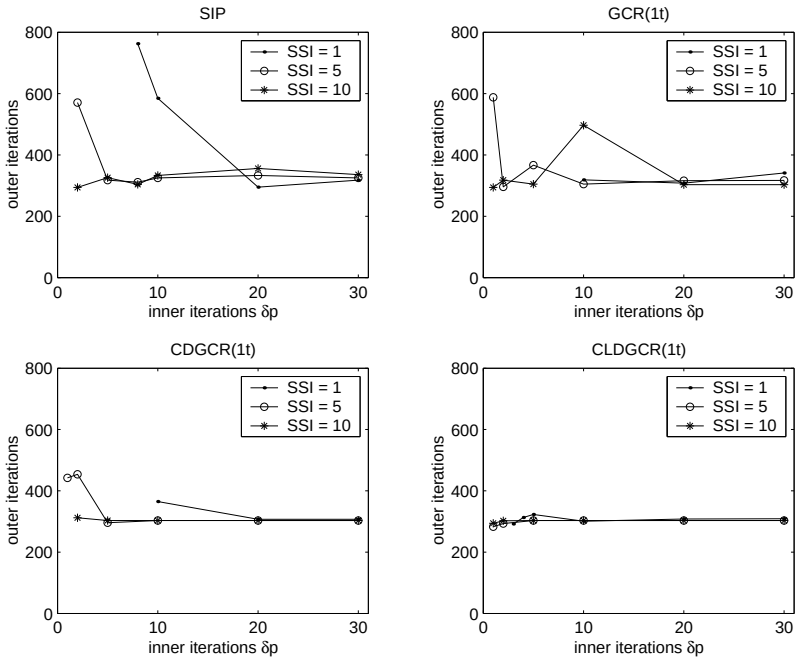

Figure 6.42: The number of outer iterations for different values of pressure-correction inner iterations and SSI iterations, comparing the SIP(1), GCR(1t), CDGCR(1t) and CLDGCR(1t) methods. Testcase III is considered.

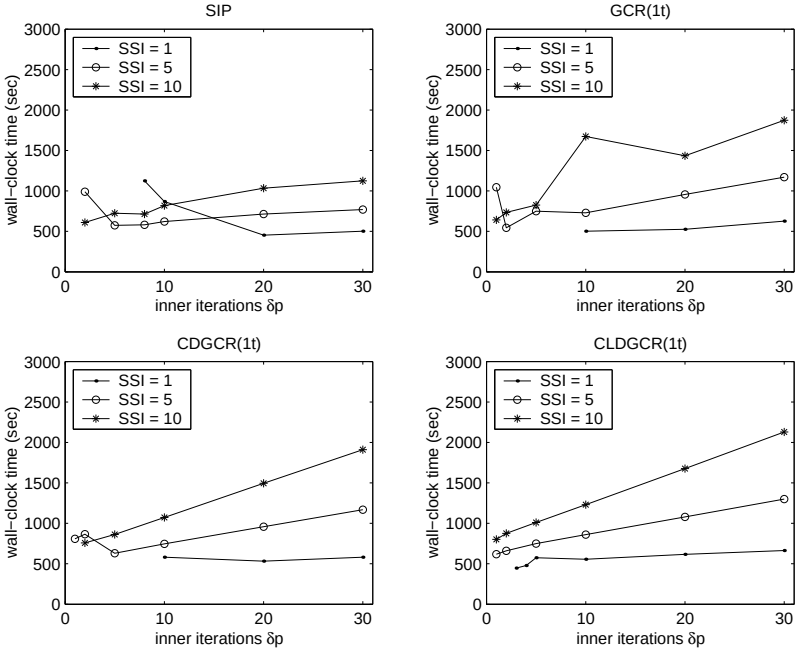

Figure 6.43: The wall-clock time for different values of pressure-correction inner iterations and SSI iterations, comparing the SIP(1), GCR(1t), CDGCR(1t) and CLDGCR(1t) methods. Testcase III is considered.

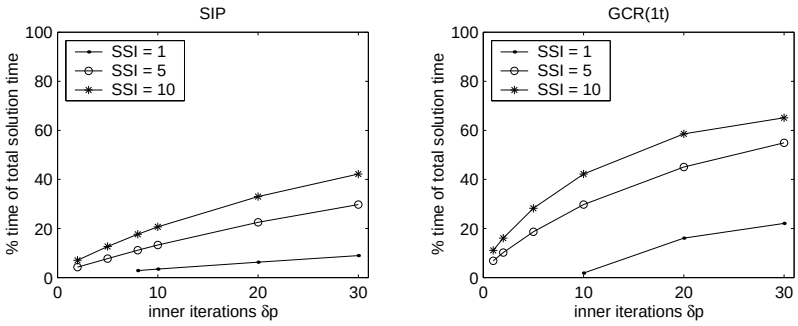

Figure 6.44: The percentage time for solving the pressure-correction system of the total wall-clock time, for the SIP(1) and the GCR(1t) methods. Testcase III is considered.

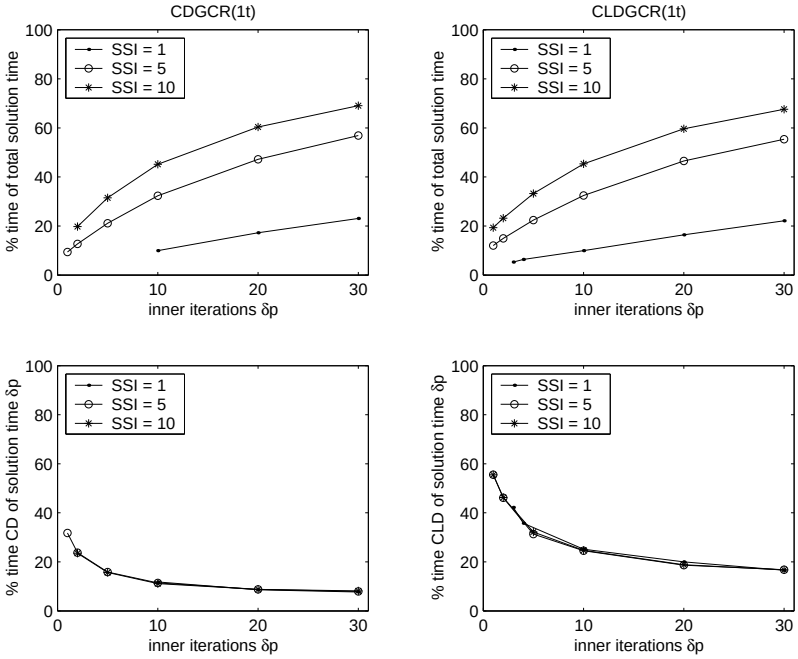

Figure 6.45: Top: the percentage time for solving the pressure-correction system of the total wall-clock time, for the CDGCR(1t) and CLDGCR(1t) methods. Bottom: the percentage time for deflation of the time for solving the pressure-correction system, for the CDGCR(1t) and the CLDGCR(1t) methods. Testcase III is considered.

7

Conclusions and recommendations

7.1 Conclusions

In this thesis, Deflated Krylov subspace methods were discussed. These methods can be used to accelerate additive Schwarz domain decomposition (DD) methods.

To gain insight into these techniques, these methods were applied to solve the Poisson equation with homogeneous Dirichlet boundary conditions (BCs) as well as homogeneous Neumann BCs. The MATLAB package was used for 1D and 2D numerical experiments. Two Krylov-Schwarz methods were implemented in MATLAB: the GCR-Schwarz and CG-Schwarz method. These methods were compared to the unaccelerated Schwarz method. Both the Krylov-Schwarz methods as the unaccelerated Schwarz method were combined with deflation. Next to constant deflation (CD), also a combination of constant with linear deflation vectors (CLD) was considered. The following conclusions can be drawn from these experiments for the Dirichlet problem:

1. The condition number for CLD deflation is better than the condition number for CD deflation.

2. From all experiments we conclude that CG-Schwarz and GCR-Schwarz perform similar. Concerning convergence rate, all the methods perform best with CLD deflation, followed by CD deflation. Concerning wall-clock times, the CLD method is more efficient for a large number of grid cells, than a small number of grid cells, per subdomain. Furthermore, it is noticeable that the unaccelerated Schwarz method improves significantly concerning convergence behavior and wall-clock time when this method is combined with deflation. For a fixed problem size, scalability is obtained for the Krylov-Schwarz methods combined with CD and CLD deflation. There seems to be an optimal decomposition for the case of CD and CLD deflation such that the wall-clock time is minimal.

3. One observation is that choosing an old search direction in the Krylov-Schwarz methods as a starting vector for the subdomain problems, results in less good convergence behavior. Another observation was made that a splitting could be found for the 1D case for CLD deflation, which gives us a bound for the spectrum of the deflation system. This seems to hold for both the unpreconditioned as the preconditioned case.

For the Neumann problem the same conclusions can be made as in conclusion 2. Furthermore, the following conclusions can be drawn from the experiments for the Neumann problem:

4. Sensitivity analysis shows that two options can be used to overcome the singularity in the deflation method due to the Neumann BCs: removing a constant deflation vector or adjusting one element in the constant deflation vector. The convergence behavior for the solution methods is similar for both options, when the second option is applied by making one element significantly larger or smaller compared to the other elements.

5. It can be observed that the unaccelerated Schwarz method behaves strangely regarding the accuracy of the subdomain solutions. When the solutions become too accurate, the convergence rate of this method decreases. Furthermore, it was observed that the splitting for the CD case can be used conform the Dirichlet problem. This seems to hold for both the unpreconditioned as for the preconditioned case.

For a comparison to the 2D case the 3D Poisson equation with homogeneous Dirichlet BCs was considered in X-stream. Both the GCR-Schwarz and CG-Schwarz methods were implemented in X-stream. Compared to the MATLAB experiments 1 iteration of the SIP method was used to solve the subdomain problems inaccurately, instead of a constant number of iterations of a basic iterative method with an standard ILU preconditioning. CD deflation and CLD deflation were combined with the Krylov-Schwarz methods. The following can be concluded:

6. Concerning convergence, it can be concluded that the unaccelerated Schwarz method performs disappointingly. The Krylov-Schwarz methods perform conform MATLAB experiments. Concerning wall-clock times, the Krylov-Schwarz methods with CLD deflation are considerably more expensive than with CD deflation.

Next to these numerical experiments to gain insight into Deflated Krylov-Schwarz methods, these techniques were used to accelerate the domain decomposition for solving the pressure-correction system in X-stream. Both the GCR-Schwarz and CG-Schwarz methods were implemented. These methods were compared to the three unaccelerated Schwarz methods, using 1 iteration of the SIP method, the SPTDMA method and the CG method to solve the subdomain problems. For the Krylov-Schwarz methods, one SIP iteration is used. Only the Krylov-Schwarz methods were combined with CD deflation and CLD deflation. The effect of restarting and truncation was investigated for the GCR-Schwarz methods. The following conclusions can be made:

7. The pressure-correction system in X-stream appears to be singular, having one zero eigenvalue and an eigenvector containing all 1's. Also, the pressure-correction seems consistent. It is not clear whether the matrix is symmetric or not.

8. Two methods can be used succesfully such that the deflation method can be applied to the singular pressure-correction system: removing one constant deflation vector or adjusting one element of the constant deflation vector. The second method appears to be the safest method for the 3D case.

9. Concerning convergence behavior, it can be concluded that the convergence behavior of the unaccelerated Schwarz methods is disappointing. The GCR-Schwarz method with deflation has the best convergence properties. Furthermore, the CG-Schwarz method performs less good compared to the GCR-Schwarz method. For the GCR-Schwarz method, it seems to be sufficient to truncate only for one search direction. Scalability

for CD and CLD deflation was shown for both Krylov-Schwarz methods. Concerning wall-clock times, we can conclude that the Krylov-Schwarz methods with deflation are in general more efficient than without deflation. Furthermore, the Krylov-Schwarz methods are competitive compared to the unaccelerated Schwarz method.

10. When an old search direction is chosen as a starting vector for the subdomain problems, this results in unfavorable convergence behavior compared to taking the zero starting vector. This is conform the MATLAB experiments. Furthermore, we observe that CLD deflation can result in a significantly large jump in the residual in the first iteration.

Since the GCR-Schwarz method with only one truncation vector has the best performance among the Krylov-Schwarz methods, this method is used to investigate the influence of solving the pressure-correction system on the total number of 'SIMPLE iterations' (outer iterations) and total wall-clock time. The effect of CD deflation and CLD deflation is investigated, and the results are compared to the default solution in X-stream, *i.e.*, the unaccelerated Schwarz method using 1 SIP iteration. Next to the number of Schwarz iterations (inner iterations), the number of SSI iterations is varied. The following can be concluded on the number of outer iterations and the total wall-clock time:

11. The number of outer iterations, for the GCR-Schwarz method with CLD deflation, is insensitive to the number of inner iterations, using CLD deflation, and the number of SSI iterations. For CD deflation a comparable effect is observed, although the method is somewhat more sensitive. Less oscillations were observed in the outer iteration residuals with deflation. For the GCR-Schwarz method without deflation and the unaccelerated Schwarz method this behavior does not occur. Therefore, we conclude that CLD deflation and CD deflation have a stabilizing effect on the outer iteration convergence behavior.

12. It was shown that a large number of outer iterations and wall-clock time can be gained by solving the pressure-correction system with a Deflated Krylov-Schwarz method.

From all the conclusions made in this section, it becomes clear that the Deflated GCR-Schwarz method is superior concerning convergence behavior and is competitive concerning wall-clock times.

7.2 Recommendations

The following recommendations can be made:

1. Concerning parallelization, first the deflation routines have to be tested with the MPI routine. Then the performance of the GCR-Schwarz and CG-Schwarz methods with deflation need to be tested in parallel. Furthermore, the RCGS orthonormalization can be implemented, which requires less global communication between processors.

2. Deflation could be applied to the energy equation as well, which is elliptic for some cases.

3. The deflation methods could be tested for non-orthogonal grids. When necessary, the deflation vectors could be constructed such that they follow the Cartesian components.

4. The GCR-SIMPLE method, which is implemented for the largest part, could be finished.

5. The additive Schwarz method in X-stream could be implemented in a slightly different way, which seems more convenient for inaccurate solution of the subdomain problems.

6. Research could be done on proving the splitting for CLD for the Dirichlet problem, and the splitting for CD for the Neumann problem. Generalization to higher order deflation vectors could be considered. Also the jump in the residual for the first iteration is left for further research.

A

PWI method

A.1 Derivation of the PWI method

In this section, the PWI method given by Equation (2.23) is derived. The basic idea of PWI is to approximate the cell face velocity components by a central discretization and adding and subtracting an extra term involving the pressure,

$$u^{\alpha}_{\boldsymbol{j}+\boldsymbol{e}_\alpha} \approx \frac{1}{2}(u^{\alpha}_{\boldsymbol{j}} + u^{\alpha}_{\boldsymbol{j}+2\boldsymbol{e}_\alpha}) + (\frac{h_1 h_2}{a^\alpha} p_{,\alpha})_{\boldsymbol{j}+\boldsymbol{e}_\alpha} - (\frac{h_1 h_2}{a^\alpha} p_{,\alpha})_{\boldsymbol{j}+\boldsymbol{e}_\alpha} \ .$$

The second and third term in the above equation are approximated differently. The second term is approximated by

$$\begin{aligned}
(\frac{h_1 h_2}{a^\alpha} p_{,\alpha})_{\boldsymbol{j}+\boldsymbol{e}_\alpha} &\approx \tfrac{1}{2}[(\frac{h_1 h_2}{a^\alpha} p_{,\alpha})_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} + (\frac{h_1 h_2}{a^\alpha} p_{,\alpha})_{\boldsymbol{j}}] \ , \\
&= \tfrac{1}{2}[(\frac{h_1 h_2}{a^\alpha})_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} (p_{,\alpha})_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} + (\frac{h_1 h_2}{a^\alpha})_{\boldsymbol{j}} (p_{,\alpha})_{\boldsymbol{j}}] \ , \\
&= \tfrac{1}{2}[(\frac{h_1 h_2}{a^\alpha})_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} \frac{p_{\boldsymbol{j}+4\boldsymbol{e}_\alpha} - p_{\boldsymbol{j}}}{2h_\alpha} + (\frac{h_1 h_2}{a^\alpha})_{\boldsymbol{j}} \frac{p_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} - p_{\boldsymbol{j}-2\boldsymbol{e}_\alpha}}{2h_\alpha}] \ , \\
&= \tfrac{1}{2}[(\frac{h_\beta}{2a^\alpha})_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} (p_{\boldsymbol{j}+4\boldsymbol{e}_\alpha} - p_{\boldsymbol{j}}) + (\frac{h_\beta}{2a^\alpha})_{\boldsymbol{j}} (p_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} - p_{\boldsymbol{j}-2\boldsymbol{e}_\alpha})] \ ,
\end{aligned} \tag{A.1}$$

and the third term by

$$\begin{aligned}
(\frac{h_1 h_2}{a^\alpha} p_{,\alpha})_{\boldsymbol{j}+\boldsymbol{e}_\alpha} &\approx (\frac{h_1 h_2}{a^\alpha})_{\boldsymbol{j}+\boldsymbol{e}_\alpha} (p_{,\alpha})_{\boldsymbol{j}+\boldsymbol{e}_\alpha} \ , \\
&= \tfrac{1}{2}[(\frac{h_1 h_2}{a^\alpha})_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} + (\frac{h_1 h_2}{a^\alpha})_{\boldsymbol{j}}] \frac{p_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} - p_{\boldsymbol{j}}}{h_\alpha} \ , \\
&= \tfrac{1}{2}[(\frac{h_\beta}{2a^\alpha})_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} (2p_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} - 2p_{\boldsymbol{j}}) + (\frac{h_\beta}{2a^\alpha})_{\boldsymbol{j}} (2p_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} - 2p_{\boldsymbol{j}})] \ .
\end{aligned} \tag{A.2}$$

Subtracting expression (A.2) from (A.1) results in

$$\begin{aligned}
&\tfrac{1}{2}[(\frac{h_\beta}{2a^\alpha})_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} (p_{\boldsymbol{j}+4\boldsymbol{e}_\alpha} - 2p_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} + p_{\boldsymbol{j}}) + (\frac{h_\beta}{2a^\alpha})_{\boldsymbol{j}} (-p_{\boldsymbol{j}+2\boldsymbol{e}_\alpha} - p_{\boldsymbol{j}-2\boldsymbol{e}_\alpha} + 2p_{\boldsymbol{j}})] \\
&\qquad = (\frac{h_\beta}{4a^\alpha} \Delta^\alpha p)|_{\boldsymbol{j}}^{\boldsymbol{j}+2\boldsymbol{e}_\alpha} \ ,
\end{aligned}$$

and by this Equation (2.23) is derived.

A.2 Discretization of the continuity equation with the PWI method

In this section, the discretised coninuity equation with the PWI method given by Equation (2.24) is derived. Writing out Equation (2.21) yields

$$h_2(u^1_{\boldsymbol{j}+\boldsymbol{e}_1} - u^1_{\boldsymbol{j}-\boldsymbol{e}_1}) + h_1(u^2_{\boldsymbol{j}+\boldsymbol{e}_2} - u^2_{\boldsymbol{j}-\boldsymbol{e}_2}) = 0 \ . \tag{A.3}$$

Approximation of the first term with the PWI method (2.23) gives

$$\begin{aligned} h_2(u^1_{\boldsymbol{j}+\boldsymbol{e}_1} - u^1_{\boldsymbol{j}-\boldsymbol{e}_1}) &= h_2([\tfrac{1}{2}(u^1_{\boldsymbol{j}} + u^1_{\boldsymbol{j}+2\boldsymbol{e}_1}) + (\frac{h_2}{4a^1}\Delta^1 p)|^{\boldsymbol{j}+2\boldsymbol{e}_1}_{\boldsymbol{j}}] \\ &\quad -[\tfrac{1}{2}(u^1_{\boldsymbol{j}-2\boldsymbol{e}_1} + u^1_{\boldsymbol{j}}) + (\frac{h_2}{4a^1}\Delta^1 p)|^{\boldsymbol{j}}_{\boldsymbol{j}-2\boldsymbol{e}_1}] \\ &= \tfrac{1}{2}h_2 u^1|^{\boldsymbol{j}+2\boldsymbol{e}_1}_{\boldsymbol{j}-2\boldsymbol{e}_1} \\ &\quad +h_2^2[(\frac{1}{4a^1}\Delta^1 p)_{\boldsymbol{j}+2\boldsymbol{e}_1} - (\frac{1}{2a^1}\Delta^1 p)_{\boldsymbol{j}} + (\frac{1}{4a^1}\Delta^1 p)_{\boldsymbol{j}-2\boldsymbol{e}_1}] \ , \end{aligned} \tag{A.4}$$

and the second term in Equation (A.3) can analogous be written as

$$h_1(u^2_{\boldsymbol{j}+\boldsymbol{e}_2} - u^2_{\boldsymbol{j}-\boldsymbol{e}_2}) = \frac{1}{2}h_1 u^2|^{\boldsymbol{j}+2\boldsymbol{e}_2}_{\boldsymbol{j}-2\boldsymbol{e}_2} + h_1^2[(\frac{1}{4a^2}\Delta^2 p)_{\boldsymbol{j}+2\boldsymbol{e}_2} - (\frac{1}{2a^2}\Delta^2 p)_{\boldsymbol{j}} + (\frac{1}{4a^2}\Delta^2 p)_{\boldsymbol{j}-2\boldsymbol{e}_2}] \ . \tag{A.5}$$

Substitution of (A.4) and (A.5) in Equation (A.3) results in

$$\begin{aligned} &\tfrac{1}{2}h_2 u^1|^{\boldsymbol{j}+2\boldsymbol{e}_1}_{\boldsymbol{j}-2\boldsymbol{e}_1} + \tfrac{1}{2}h_1 u^2|^{\boldsymbol{j}+2\boldsymbol{e}_2}_{\boldsymbol{j}-2\boldsymbol{e}_2} \\ &+ h_2^2[(\frac{1}{4a^1}\Delta^1 p)_{\boldsymbol{j}+2\boldsymbol{e}_1} - (\frac{1}{2a^1}\Delta^1 p)_{\boldsymbol{j}} + (\frac{1}{4a^1}\Delta^1 p)_{\boldsymbol{j}-2\boldsymbol{e}_1}] \\ &+ h_1^2[(\frac{1}{4a^2}\Delta^2 p)_{\boldsymbol{j}+2\boldsymbol{e}_2} - (\frac{1}{2a^2}\Delta^2 p)_{\boldsymbol{j}} + (\frac{1}{4a^2}\Delta^2 p)_{\boldsymbol{j}-2\boldsymbol{e}_2}] = 0 \ . \end{aligned}$$

When this equation is multiplied by a factor 2, this results in (2.24).

B

RCGS algorithm and LU factorization algorithms

Algorithm B.1 (RCGS). Given the vectors $\boldsymbol{s}^{(k)}$ and $\boldsymbol{v}^{(k)}$, the sets $\{\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(k-1)}\}$ and $\{\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(k-1)}\}$, this algorithm computes an orthonormal basis for $\{\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(k)}\}$, preserving the relation $A\boldsymbol{s}^{(k)} = \boldsymbol{v}^{(k)}$.

for $m = 1, 2$ **do**
  $\beta = (\boldsymbol{v}^{(k)}, \boldsymbol{v}^{(k)})$;
  for $i = 1, \ldots, k-1$ **do**
    $\alpha_i = (\boldsymbol{v}, \boldsymbol{v}^{(j)})$;
  end for
  $\beta := (\beta - \sum_{i=1}^{k-1} \alpha_i^2)^{\frac{1}{2}}$;
  $\boldsymbol{v}^{(k)} := \beta^{-1}(\boldsymbol{v} - \sum_{i=1}^{k-1} \alpha_i \boldsymbol{v}^{(i)})$;
  $\boldsymbol{s}^{(k)} := \beta^{-1}(\boldsymbol{s} - \sum_{i=1}^{k-1} \alpha_i \boldsymbol{s}^{(i)})$;
end for
$\boldsymbol{v}^{(k)} := \boldsymbol{v}^{(k)} / \|\boldsymbol{v}^{(k)}\|_2$;
$\boldsymbol{s}^{(k)} := \boldsymbol{s}^{(k)} / \|\boldsymbol{s}^{(k)}\|_2$;

Algorithm B.2 (LU decomposition). Given an $n \times n$ matrix A, and the $n \times n$ matrices L and U, containing only zero coefficients. Then this algorithm computes an LU factorization of A.

for $k = 1, \ldots, n-1$ **do**
  for $i = k+1, \ldots, n$ **do**
    $L(i,k) = A(i,k)/A(k,k)$;
  end for
  for $j = k+1, \ldots, n$ **do**
    for $i = k+1, \ldots, n$ **do**
      $U(i,j) = A(i,j) - L(i,k)A(k,j)$;
    end for
  end for
  $L(k,k) = 1$;
end for
$L(n,n) = 1$;

Algorithm B.3 (Forward-substitution). Given an $n \times n$ matrix L which is unit lower triangular, and a vector $\boldsymbol{b}$. Then this algorithm solves the system $L\boldsymbol{y} = \boldsymbol{b}$.

for $i = 1, \ldots, n$ **do**

```
    for j = 1, ..., i − 1 do
        y(i) = b(i) − L(i, j)y(j);
    end for
end for
```

Algorithm B.4 (Back-substitution). Given an $n \times n$ matrix U, which is upper triangular, and a vector $\boldsymbol{b}$. Then this algorithm solves the system $U\boldsymbol{y} = \boldsymbol{b}$.

```
for i = n, ..., 1 do
    for j = i + 1, ..., n do
        y(i) = ({b(i) − U(i, j)y(j)}/U(i, i);
    end for
end for
```

C

GCR-SIMPLE method on a colocated grid

The GCR-SIMPLE method for a colocated grid arrangement can be obtained in a similar way as for the staggered case, see Vuik & Saghir [31].

In this paper, a diagonal scaling of the system (3.29) is considered, mainly because the Dirichlet BCs for the velocities were implemented in a rather untidy way. A condition $u_{\mathrm{P}} = g_{\mathrm{P}}$ in a point P was imposed by adding a large real number $c_{\max}$ to the main diagonal entry of the coefficient matrix, and adding $c_{\max} g_{\mathrm{P}}$ to the RHS vector. This had a negative effect on the solvers used to solve the systems in the GCR-SIMPLE method. With a diagonal scaling, much better results were observed, also due to the fact that diagonal scaling leads to a better behavior with respect to rounding errors.

With diagonal scaling, the postconditioned system (3.35) is replaced by

$$D_{AB}^{-1} AB\hat{\boldsymbol{y}} = D_{AB}^{-1}\boldsymbol{b} \ , \quad \boldsymbol{y} = B\hat{\boldsymbol{y}} \ , \tag{C.1}$$

where $D_{AB}^{-1} \equiv \mathrm{diag}(AB)^{-1}$. With the splitting $D_{AB}^{-1}AB = D_{AB}^{-1}M - D_{AB}^{-1}N$, applying right-preconditioning to this system results in:

$$(D_{AB}^{-1}A)(D_{AB}^{-1}MB^{-1})^{-1}\bar{\boldsymbol{y}} = D_{AB}^{-1}\boldsymbol{b} \ , \quad \boldsymbol{y} = (D_{AB}^{-1}MB^{-1})^{-1}\bar{\boldsymbol{y}} \ . \tag{C.2}$$

This system is of the form (3.20), and therefore, the GCR-SIMPLE algorithm with scaling is obtained by taking $A \equiv D_{AB}^{-1}A$, $\boldsymbol{b} \equiv D_{AB}^{-1}\boldsymbol{b}$, and $M \equiv D_{AB}^{-1}MB^{-1}$ in Algorithm (3.1), which results into the following algorithm.

Algorithm C.1 (GCR-SIMPLE method with diagonal scaling). Let the matrix A be given by (3.34), and the vector $\boldsymbol{b}$ as in (3.5), and let B and M be given by (3.33). Let $\boldsymbol{y}^{(0)}$ be an initial solution, such that $A\boldsymbol{y}^{(0)} \approx \boldsymbol{b}$. Then this algorithms solves $A\boldsymbol{y} = \boldsymbol{b}$.

$\boldsymbol{r}^{(0)} = D_{AB}^{-1}(\boldsymbol{b} - A\boldsymbol{y}^{(0)})$;
for $k = 1, \ldots,$ ngcrsimple **do**
  Solve $D_{AB}^{-1}M\boldsymbol{q} = \boldsymbol{r}^{(k-1)}$;
  $\boldsymbol{s}^{(k)} = B\boldsymbol{q}$;
  $\boldsymbol{v}^{(k)} = D_{AB}^{-1}A\boldsymbol{s}^{(k)}$;
  Orthonormalize $\boldsymbol{s}^{(k)}$ and $\boldsymbol{v}^{(k)}$ by, for example, MGS (Algorithm 3.2);
  $\beta = (\boldsymbol{r}^{(k-1)}, \boldsymbol{v}^{(k)})$;
  $\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + \beta\boldsymbol{s}^{(k)}$;
  $\boldsymbol{r}^{(k)} = \boldsymbol{r}^{(k-1)} - \beta\boldsymbol{v}^{(k)}$;
end for
$\boldsymbol{y} \approx \boldsymbol{y}^{(k)}$;

In this algorithm $\boldsymbol{q}$ is an auxiliary variable. Diagonal scaling for the colocated case is not so obvious, as we will illustrate now. Consider

$$AB = \begin{bmatrix} Q_1 & 0 & G_1 - Q_1 D_1^{-1} G_1 \\ 0 & Q_2 & G_2 - Q_2 D_2^{-1} G_2 \\ G_1^T & G_2^T & \hat{R} \end{bmatrix} ,$$

where $\hat{R} = R$ in the staggered case and $\hat{R} = R + C$ in the colocated case, with R given by $R = -\sum_{i=1}^{2} G_i^T D_i^{-1} G_i$. Since $D_1 = \mathrm{diag}(Q_1)$ and $D_2 = \mathrm{diag}(Q_2)$ in the SIMPLE method, and we therefore should take

$$D_{AB} = \begin{bmatrix} D_1 & 0 & 0 \\ 0 & D_2 & 0 \\ 0 & 0 & D_3 \end{bmatrix} ,$$

with $D_3 \equiv \mathrm{diag}(R)$ for the staggered case and $D_3 \equiv \mathrm{diag}(C + R)$ for the colocated case. The problem for the colocated grid is that the matrix C is not explicitly available. Therefore, one could choose $D_{AB} = \mathrm{diag}(R)$, just like in the staggered case. However, when $\|C\|$ is not negligible compared to $\|R\|$, it is not known what effect this shall have on the performance of the GCR-SIMPLE algorithm.

The following algorithm is a detailed version of Algorithm C.1. The matrices denoted by $\boxed{}$ are the contributions to the diagonal scaling. Removing those matrices, results in the GCR-SIMPLE algorithm without diagonal scaling.

Algorithm C.2 (GCR-SIMPLE method with diagonal scaling (detailed)). Let the matrix A be given by (3.34), and the vector $\boldsymbol{b}$ as in (3.29), and let Q_i and R and D_i be given as in (3.33). Let $\boldsymbol{y}^{(0)}$ be an initial solution, such that $A\boldsymbol{y}^{(0)} \approx \boldsymbol{b}$. Then this algorithm solves $A\boldsymbol{y} = \boldsymbol{b}$.

$\boldsymbol{r}_i^{(0)} = \boxed{D_i^{-1}} (\boldsymbol{b}_i - G_i \boldsymbol{p}^{(0)} - Q_i \boldsymbol{u}_i^{(0)}), \quad i = 1, \ldots, d;$
$\boldsymbol{r}_{d+1}^{(0)} = \boxed{D_{d+1}^{-1}} (\boldsymbol{b}_{d+1} - \sum_{i=1}^{d} G_i^T \boldsymbol{u}_i^{(0)} - C\boldsymbol{p}^{(0)});$
for $k = 1, \ldots,$ ngcrsimple **do**
  Solve $\boxed{D_i^{-1}}\, Q_i \boldsymbol{q}_i = \boldsymbol{r}_i^{(k-1)}, \quad i = 1, \ldots, d;$
  Solve $\boxed{D_{d+1}^{-1}}\, R\boldsymbol{q}_{d+1} = \boldsymbol{r}_{d+1}^{(k-1)} - \boxed{D_{d+1}^{-1}} \sum_{i=1}^{d} G_i^T \boldsymbol{q}_i;$
  $\boldsymbol{s}_i^{(k)} = \boldsymbol{q}_i - D_i^{-1} G_i \boldsymbol{q}_{d+1}, \quad i = 1, \ldots, d;$
  $\boldsymbol{s}_{d+1}^{(k)} = \boldsymbol{q}_{d+1};$
  $\boldsymbol{v}_i^{(k)} = \boxed{D_i^{-1}} (Q_i \boldsymbol{s}_i^{(k)} + G_i \boldsymbol{s}_{d+1}^{(k)}), \quad i = 1, \ldots, d;$
  $\boldsymbol{v}_{d+1}^{(k)} = \boxed{D_{d+1}^{-1}} (\sum_{i=1}^{d} G_i^T \boldsymbol{s}_i^{(k)} + C\boldsymbol{s}_{d+1}^{(k)});$
  Orthonormalize $\boldsymbol{s}^{(k)}$ and $\boldsymbol{v}^{(k)}$ by, for example, MGS (Algorithm 3.2);
  $\beta = (\boldsymbol{r}^{(k-1)}, \boldsymbol{v}^{(k)});$
  $\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)} + \beta \boldsymbol{s}^{(k)};$
  $\boldsymbol{r}^{(k)} = \boldsymbol{r}^{(k-1)} - \beta \boldsymbol{v}^{(k)};$
end for
$\boldsymbol{y} \approx \boldsymbol{y}^{(k)};$

It is shown in Vuik & Saghir [31] 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. In this paper, it is also stated that for good performance of the GCR-SIMPLE(R) method a pressure BC should be chosen at an outlet.

During the Master's project, the largest part of the implementation of the GCR-SIMPLE method in the X-stream code is done. For making a new method to work, it is always advisable to start with an elementary problem. A suitable setup is given by Testcase I (see Subsection 6.3.1), with the unit cube divided into a small number of cells (*e.g.* 2 cells in each dimension). For this testcase, it turns out that the pressure is smooth enough and no PWI is necessary to obtain a correct checkerboard-free solution. A problem to overcome in the X-stream code seems to be the treatment of the pressure. 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.

D

Figures and tables numerical experiments

D.1 Figures

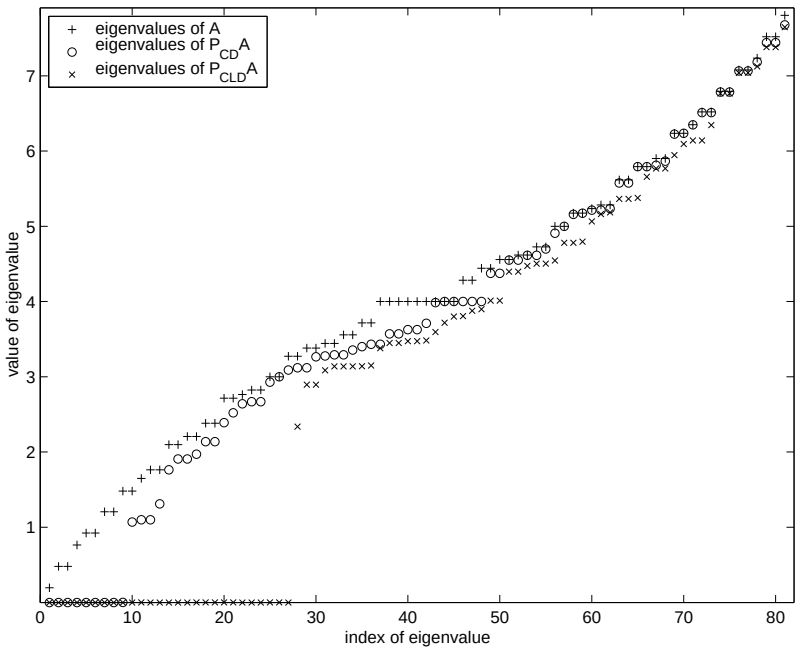

Figure D.1: Eigenvalue spectra for the case of no deflation, CD deflation, and CLD deflation. The Dirichlet problem is considered for a 9×9 grid and 3×3 subdomains, and the unpreconditioned case is considered.

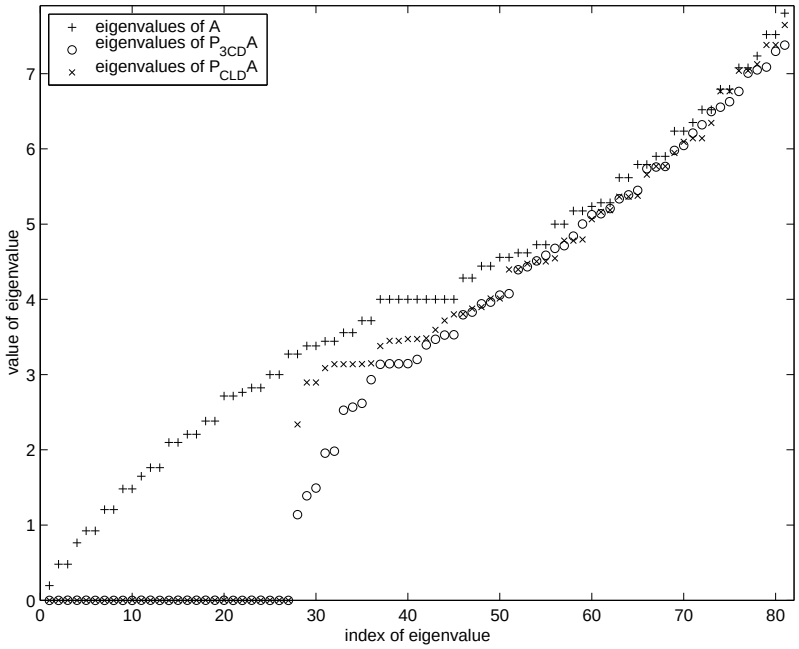

Figure D.2: Eigenvalue spectra for the case of no deflation, CLD deflation, and 3CD deflation. The Dirichlet problem is considered for a 9×9 grid and 3×3 subdomains, and the unpreconditioned case is considered.

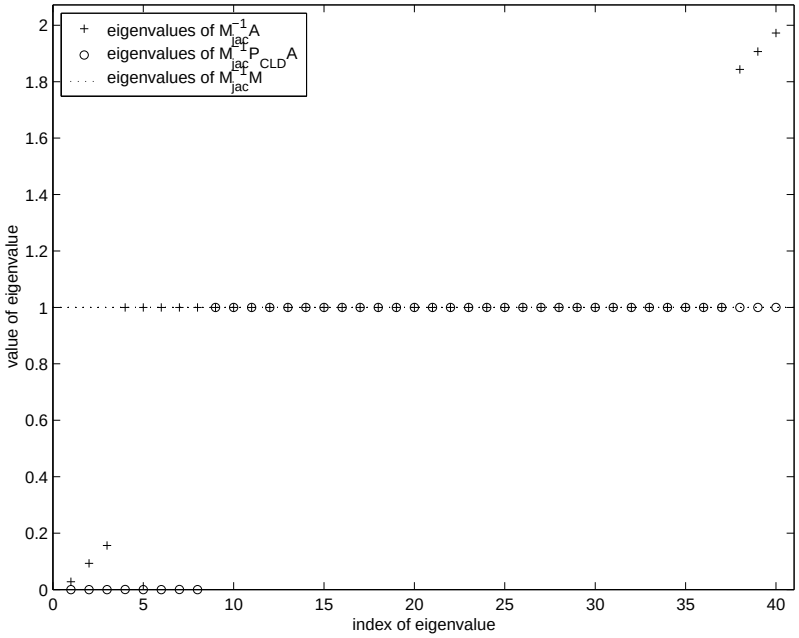

Figure D.3: Eigenvalue spectra for the splitting for the case of CLD deflation. The 1D Dirichlet problem is considered for 4 subdomains and 10 grid cells per subdomain, and the preconditioned case is considered.

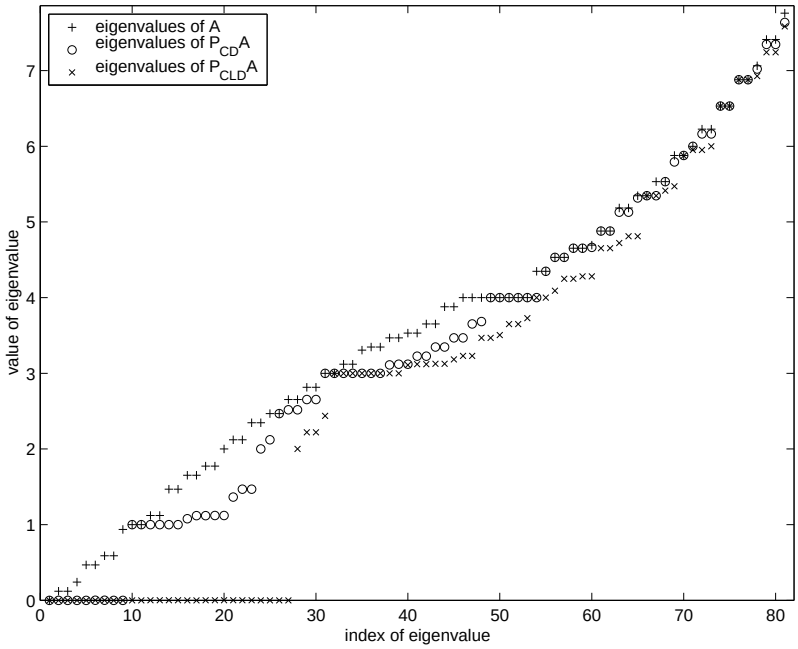

Figure D.4: Eigenvalue spectra for the case of no deflation, CD deflation, and CLD deflation. The Neumann problem is considered for a 9×9 grid and 3×3 subdomains, and the unpreconditioned case is considered.

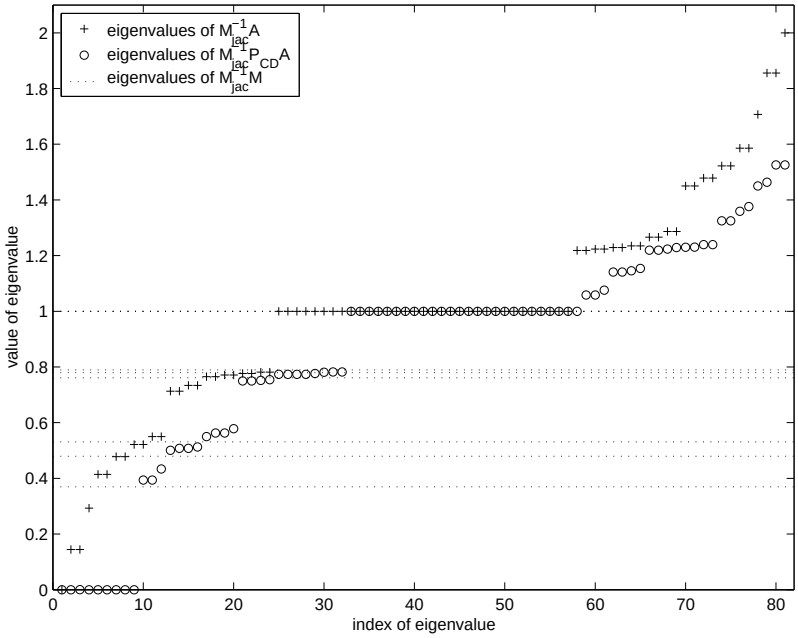

Figure D.5: Eigenvalue spectra for the same splitting used for the Dirichlet case, but now for the Neumann case. The preconditioned case is considered.

D.2 Tables

Method	*S.d. iter.*	2 × 2	3 × 3	4 × 4	5 × 5
GCR	15	30 (285.06)	36 (239.36)	42 (237.66)	47 (254.86)
CG	15	32 (309.39)	37 (248.27)	44 (248.65)	48 (259.14)
ILU	15	– (–.–)	– (–.–)	– (–.–)	– (–.–)
CDGCR	15	28 (269.85)	28 (189.93)	27 (157.31)	25 (141.88)
CDCG	15	28 (275.94)	30 (206.79)	28 (166.15)	26 (150.55)
CDILU	15	182 (1648.10)	163 (1063.08)	110 (619.12)	98 (532.94)
CLDGCR	15	23 (227.07)	22 (157.25)	21 (135.33)	20 (136.22)
CLDCG	15	23 (234.20)	23 (168.13)	22 (144.78)	19 (134.62)
CLDILU	15	83 (764.34)	72 (485.27)	64 (381.07)	54 (326.76)
GCR	5	32 (206.80)	38 (138.14)	43 (113.15)	49 (116.33)
CG	5	33 (215.97)	43 (154.01)	45 (114.84)	49 (111.06)
ILU	5	– (–.–)	– (–.–)	– (–.–)	– (–.–)
CDGCR	5	29 (190.53)	31 (116.34)	28 (77.99)	25 (65.47)
CDCG	5	30 (199.82)	32 (120.45)	28 (77.89)	25 (65.53)
CDILU	5	236 (1417.90)	180 (630.54)	127 (326.92)	101 (238.76)
CLDGCR	5	24 (163.51)	22 (90.41)	21 (71.46)	20 (75.09)
CLDCG	5	24 (167.58)	23 (95.79)	22 (75.14)	20 (76.02)
CLDILU	5	108 (660.76)	85 (313.95)	72 (207.24)	47 (143.57)
GCR	2	41 (225.91)	46 (125.89)	50 (86.45)	52 (76.01)
GCR(5t)	2	59 (305.44)	59 (234.92)	64 (101.78)	71 (94.31)
GCR(5r)	2	113 (578.80)	84 (223.33)	121 (190.71)	110 (145.17)
CG	2	44 (241.12)	46 (121.88)	52 (83.44)	54 (71.35)
ILU	2	– (–.–)	– (–.–)	– (–.–)	– (–.–)
CDGCR	2	34 (190.42)	32 (90.51)	30 (55.80)	27 (45.26)
CDGCR(5t)	2	41 (217.23)	36 (100.68)	30 (53.66)	27 (44.00)
CDGCR(5r)	2	75 (389.21)	46 (126.99)	36 (63.27)	33 (52.01)
CDCG	2	38 (212.62)	36 (99.96)	31 (56.02)	27 (44.49)
CDILU	2	– (–.–)	205 (544.81)	149 (242.93)	126 (178.29)
CLDGCR	2	26 (151.87)	24 (75.68)	22 (54.05)	21 (58.27)
CLDGCR(5t)	2	28 (155.99)	26 (81.02)	22 (53.17)	21 (57.67)
CLDGCR(5r)	2	34 (186.25)	31 (94.25)	27 (61.79)	23 (60.80)
CLDCG	2	28 (165.17)	25 (78.98)	23 (55.80)	21 (58.06)
CLDILU	2	136 (704.42)	75 (210.80)	77 (148.41)	50 (103.65)

Table D.1: Iterative solution methods compared for a fixed problem size of a 60×60 grid, for the Dirichlet problem. Four decompositions are considered and the subdomain solutions are obtained by 15, 5, and 2 iterations with a ILU BIM. Iterations are given and wall-clock times are in parentheses (sec.) A '–' denotes that the method did no convergence in less than 250 iterations.

Method	*S.d. iter.*	2×2	3×3	4×4	5×5
GCR	15	33 (314.50)	42 (280.83)	48 (272.98)	53 (290.07)
CG	15	35 (338.17)	42 (282.37)	49 (276.93)	54 (291.38)
ILU	15	(–.–)	(–.–)	(–.–)	(–.)
CDGCR	15	28 (269.36)	31 (209.87)	29 (168.62)	26 (147.25)
CDCG	15	29 (284.79)	32 (220.58)	29 (171.15)	27 (155.72)
CDILU	15	161 (1481.13)	151 (975.31)	118 (654.66)	102 (546.35)
CLDGCR	15	23 (226.96)	23 (163.46)	21 (134.73)	20 (136.77)
CLDCG	15	22 (224.87)	22 (161.97)	21 (138.75)	19 (134.46)
CLDILU	15	84 (783.78)	57 (381.29)	69 (402.44)	49 (294.02)
GCR	5	41 (264.85)	46 (168.40)	50 (132.98)	56 (134.35)
CG	5	42 (271.88)	54 (193.33)	54 (137.48)	56 (126.82)
ILU	5	(–.–)	(–.–)	(–.–)	(–.)
CDGCR	5	34 (222.23)	34 (127.10)	30 (83.05)	26 (67.83)
CDCG	5	37 (243.10)	34 (127.70)	30 (82.72)	27 (70.14)
CDILU	5	212 (1312.72)	183 (634.11)	125 (316.42)	104 (242.00)
CLDGCR	5	26 (175.88)	23 (93.71)	21 (70.63)	20 (74.75)
CLDCG	5	25 (173.57)	23 (95.79)	21 (71.68)	21 (78.22)
CLDILU	5	118 (741.61)	69 (252.11)	65 (184.49)	58 (168.01)
GCR	2	57 (314.72)	56 (154.61)	62 (109.19)	62 (92.44)
GCR(5t)	2	70 (375.03)	82 (222.88)	89 (139.01)	90 (117.43)
GCR(5r)	2	124 (659.27)	125 (334.06)	141 (219.70)	202 (261.72)
CG	2	66 (356.96)	68 (178.01)	72 (114.52)	76 (99.80)
ILU	2	– (–.–)	– (–.–)	– (–.–)	– (–.)
CDGCR	2	43 (239.42)	35 (98.87)	32 (58.56)	28 (46.63)
CDGCR(5t)	2	46 (251.11)	37 (101.17)	32 (55.97)	28 (44.49)
CDGCR(5r)	2	66 (355.81)	51 (137.37)	42 (71.62)	36 (55.15)
CDCG	2	47 (259.91)	39 (108.92)	34 (60.26)	29 (46.69)
CDILU	2	– (–.–)	231 (604.01)	148 (238.32)	125 (174.67)
CLDGCR	2	30 (173.24)	25 (77.89)	23 (54.82)	20 (56.02)
CLDGCR(5t)	2	34 (192.68)	26 (79.04)	23 (53.39)	20 (55.03)
CLDGCR(5r)	2	45 (250.63)	33 (97.38)	27 (60.36)	23 (59.71)
CLDCG	2	31 (180.71)	27 (83.76)	24 (56.30)	21 (56.02)
CLDILU	2	191 (1019.58)	111 (300.44)	60 (116.17)	59 (116.11)

Table D.2: Iterative solution methods compared for a fixed problem size of a 60×60 grid, for the Neumann problem. Four decompositions are considered and the subdomain solutions are obtained by 15, 5, and 2 iterations with a ILU BIM. Iterations are given and wall-clock times are in parentheses (sec.). A '–' in the table denotes that the method did no convergence in less than 250 iterations.

E

Figures and tables results in X-stream

E.1 Figures

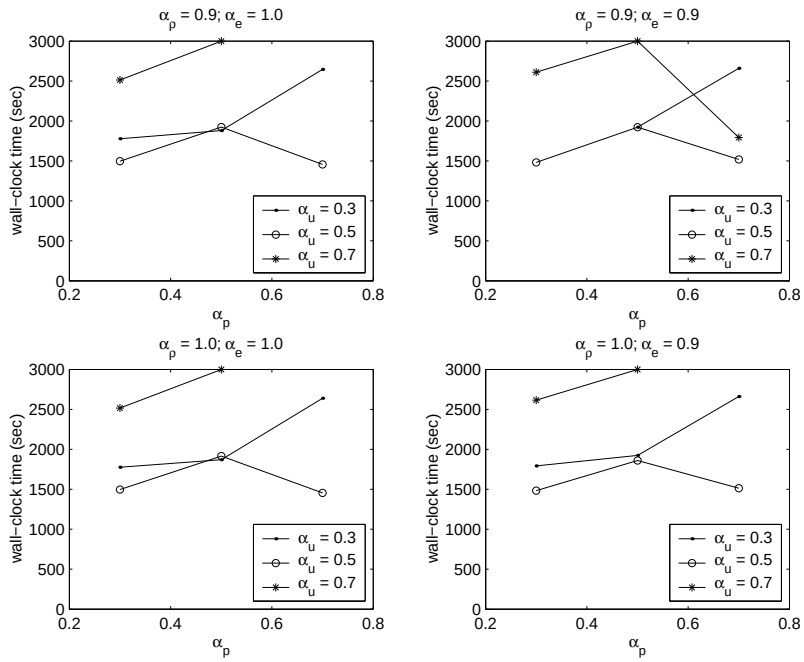

Figure E.1: The influence of parameter variation on the wall-clock times. Testcase II is considered for a 4×4 decompositon.

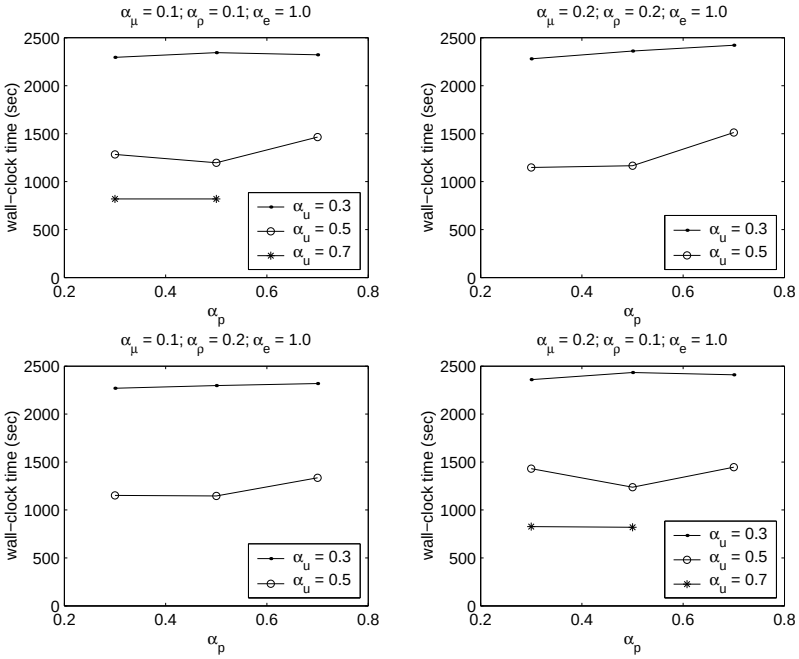

Figure E.2: The influence of parameter variation on the wall-clock times. Testcase III is considered.

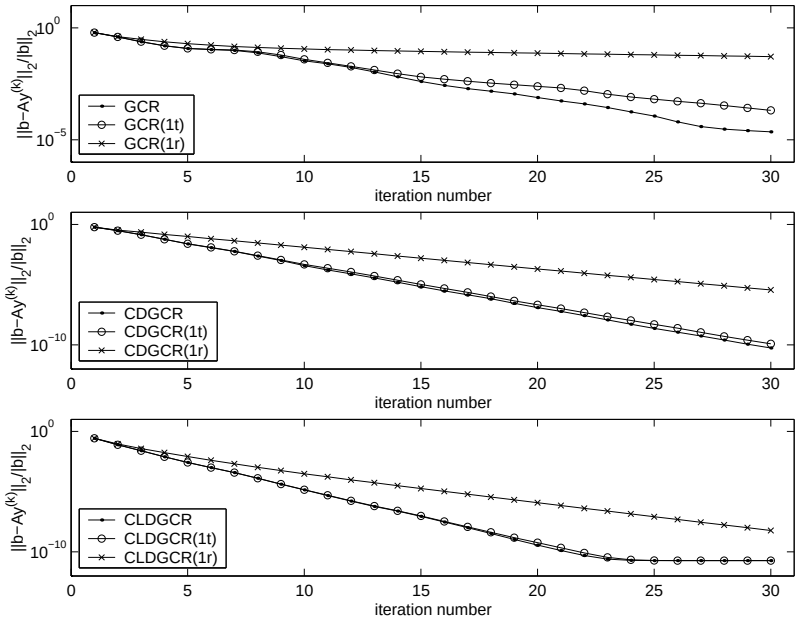

Figure E.3: GCR-Schwarz methods compared using restarting and truncation for only one search direction, for solving the pressure-correction system. Testcase I is considered for a $4 \times 4 \times 4$ decompositon.

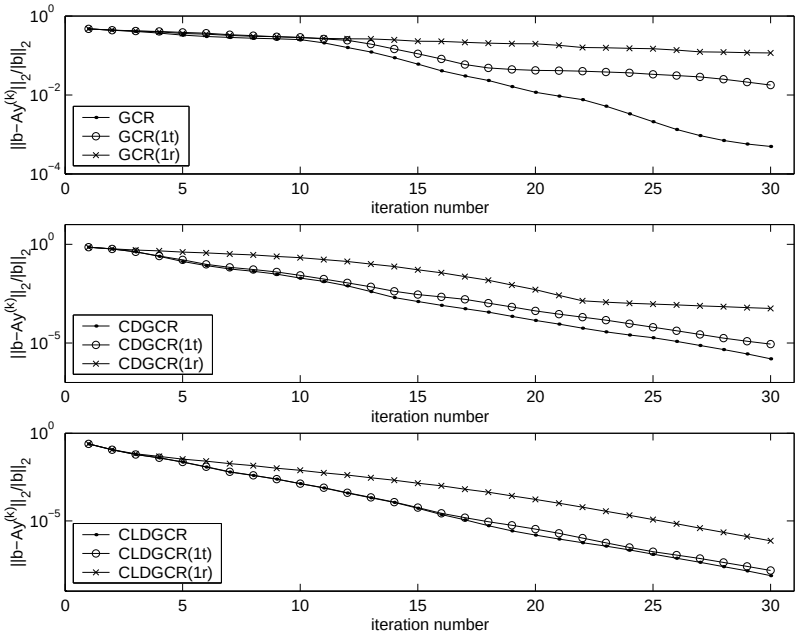

Figure E.4: GCR-Schwarz methods compared using restarting and truncation for only one search direction, for solving the pressure-correction system. Testcase II is considered for a 4×4 decompositon.

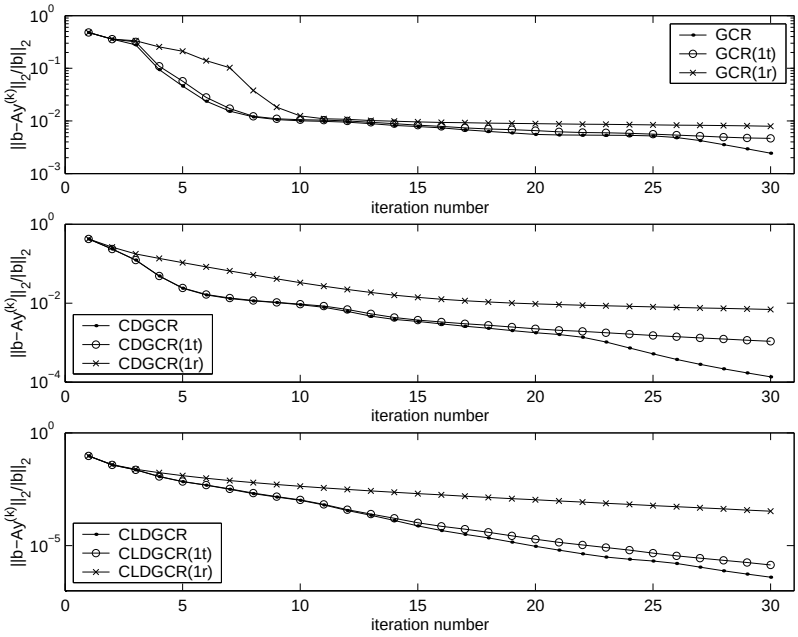

Figure E.5: GCR-Schwarz methods compared using restarting and truncating for only one search direction, for solving the pressure-correction system. Testcase III is considered.

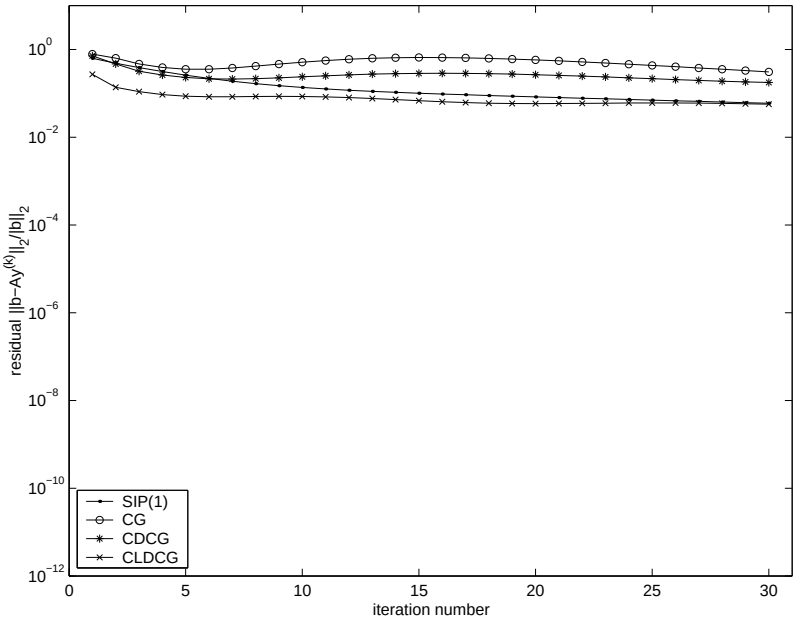

Figure E.6: Convergence behavior of the CG-Schwarz method, when an old search direction is taken as a starting vector. Testcase I is considered for a $4 \times 4 \times 4$ decompositon.

E.2 Tables

Method	crit. 1 iter.	crit. 30 iter.
SIP(1)	0.86	$1.20 \cdot 10^{-1}$
CG(1)	1.51	$1.18 \cdot 10^{-3}$
SPTDMA(1)	0.86	$9.50 \cdot 10^{-1}$
CG	1.40	$2.02 \cdot 10^{-5}$
CDCG	0.72	$6.02 \cdot 10^{-10}$
CLDCG	0.28	$2.37 \cdot 10^{-12}$
GCR	0.86	$9.84 \cdot 10^{-7}$
CDGCR	0.59	$1.87 \cdot 10^{-11}$
CLDGCR	0.27	$2.09 \cdot 10^{-14}$
GCR(1r)	0.86	$9.38 \cdot 10^{-2}$
CDGCR(1r)	0.59	$3.34 \cdot 10^{-5}$
CLDGCR(1r)	0.27	$9.52 \cdot 10^{-9}$
GCR(1t)	0.86	$7.94 \cdot 10^{-6}$
CDGCR(1t)	0.59	$2.99 \cdot 10^{-11}$
CLDGCR(1t)	0.27	$2.36 \cdot 10^{-14}$
GCR(3r)	0.86	$2.03 \cdot 10^{-4}$
CDGCR(3r)	0.59	$2.58 \cdot 10^{-11}$
CLDGCR(3r)	0.27	$2.11 \cdot 10^{-14}$
GCR(3t)	0.86	$2.71 \cdot 10^{-5}$
CDGCR(3t)	0.59	$1.87 \cdot 10^{-11}$
CLDGCR(3t)	0.27	$2.11 \cdot 10^{-14}$

Method	10 iter. SIP(1)	20 iter. SIP(1)
SIP(1)	10 (2.70)	20 (5.11)
CG(1)	10 (2.70)	12 (3.30)
SPTDMA(1)	– (–.–)	– (–.–)
CG	6 (5.61)	8 (7.11)
CDCG	2 (3.30)	3 (4.11)
CLDCG	1 (19.90)	2 (21.20)
GCR	4 (3.71)	6 (5.51)
CDGCR	2 (2.80)	3 (3.51)
CLDGCR	1 (19.50)	2 (20.70)
GCR(1r)	8 (6.61)	18 (14.40)
CDGCR(1r)	2 (2.80)	4 (4.41)
CLDGCR(1r)	1 (19.50)	2 (20.70)
GCR(1t)	5 (4.51)	6 (5.21)
CDGCR(1t)	2 (2.80)	3 (3.61)
CLDGCR(1t)	1 (19.50)	2 (20.80)
GCR(3r)	6 (5.31)	7 (6.01)
CDGCR(3r)	2 (2.80)	3 (3.51)
CLDGCR(3r)	1 (19.50)	2 (20.70)
GCR(3t)	4 (3.81)	6 (5.31)
CDGCR(3t)	2 (2.80)	3 (3.51)
CLDGCR(3t)	1 (19.50)	2 (20.70)

Table E.1: Left table: absolute termination criteria of the solution methods after 1 and 30 iterations, for solving the 3D Poisson equation. Right table: iterations required and wall-clock time in parentheses ($\cdot 10^{-1}$ sec.), to obtain the same criterion as the SIP(1) method has after 10 and 20 iterations.

	$1 \times 1 \times 1$		$2 \times 2 \times 2$		$4 \times 4 \times 4$	
Method	crit. 1 iter.	crit. 30 iter.	crit. 1 iter.	crit. 30 iter.	crit. 1 iter.	crit. 30 iter.
SIP(1)	0.35	$4.86 \cdot 10^{-3}$	0.46	$1.80 \cdot 10^{-2}$	0.63	$5.93 \cdot 10^{-2}$
CG(1)	0.91	$1.25 \cdot 10^{-2}$	0.81	$1.28 \cdot 10^{-2}$	0.88	$1.03 \cdot 10^{-2}$
SPTDMA(1)	0.54	$9.90 \cdot 10^{-2}$	0.54	$1.90 \cdot 10^{-1}$	0.65	$5.30 \cdot 10^{-1}$
CG	0.56	$2.51 \cdot 10^{-8}$	0.54	$1.05 \cdot 10^{-4}$	0.78	$4.26 \cdot 10^{-3}$
CDCG	0.56	$2.51 \cdot 10^{-8}$	0.57	$2.36 \cdot 10^{-6}$	0.71	$1.48 \cdot 10^{-7}$
CLDCG	0.33	$7.49 \cdot 10^{-9}$	0.46	$2.16 \cdot 10^{-7}$	0.27	$4.75 \cdot 10^{-10}$
GCR	0.34	$1.67 \cdot 10^{-12}$	0.45	$3.30 \cdot 10^{-7}$	0.61	$2.28 \cdot 10^{-5}$
CDGCR	0.34	$1.67 \cdot 10^{-12}$	0.46	$2.41 \cdot 10^{-9}$	0.59	$5.58 \cdot 10^{-11}$
CLDGCR	0.26	$2.64 \cdot 10^{-13}$	0.42	$1.16 \cdot 10^{-11}$	0.26	$1.86 \cdot 10^{-11}$
GCR(1r)	0.34	$6.55 \cdot 10^{-5}$	0.45	$8.38 \cdot 10^{-5}$	0.61	$5.21 \cdot 10^{-4}$
CDGCR(1r)	0.34	$6.55 \cdot 10^{-5}$	0.46	$1.07 \cdot 10^{-4}$	0.59	$3.63 \cdot 10^{-6}$
CLDGCR(1r)	0.26	$7.78 \cdot 10^{-7}$	0.42	$8.17 \cdot 10^{-6}$	0.26	$5.94 \cdot 10^{-9}$
GCR(1t)	0.34	$2.65 \cdot 10^{-9}$	0.45	$4.24 \cdot 10^{-6}$	0.61	$2.06 \cdot 10^{-4}$
CDGCR(1t)	0.34	$2.65 \cdot 10^{-9}$	0.46	$1.52 \cdot 10^{-8}$	0.59	$1.25 \cdot 10^{-10}$
CLDGCR(1t)	0.26	$5.93 \cdot 10^{-12}$	0.42	$1.28 \cdot 10^{-10}$	0.26	$1.87 \cdot 10^{-11}$
GCR(3r)	0.34	$7.25 \cdot 10^{-8}$	0.45	$1.26 \cdot 10^{-4}$	0.61	$8.80 \cdot 10^{-3}$
CDGCR(3r)	0.34	$7.25 \cdot 10^{-8}$	0.46	$8.11 \cdot 10^{-7}$	0.59	$7.36 \cdot 10^{-9}$
CLDGCR(3r)	0.26	$1.03 \cdot 10^{-10}$	0.42	$4.33 \cdot 10^{-9}$	0.26	$1.87 \cdot 10^{-11}$
GCR(3t)	0.34	$2.02 \cdot 10^{-10}$	0.45	$3.21 \cdot 10^{-6}$	0.61	$6.27 \cdot 10^{-5}$
CDGCR(3t)	0.34	$2.02 \cdot 10^{-10}$	0.46	$1.15 \cdot 10^{-8}$	0.59	$6.33 \cdot 10^{-11}$
CLDGCR(3t)	0.26	$6.63 \cdot 10^{-12}$	0.42	$6.14 \cdot 10^{-11}$	0.26	$1.87 \cdot 10^{-11}$

Table E.2: Absolute termination criteria of the solution methods after 1 and 30 iterations, for solving the pressure-correction system. Testcase I is considered.

	$1 \times 1 \times 1$		$2 \times 2 \times 2$		$4 \times 4 \times 4$	
Method	10 iter. SIP(1)	20 iter. SIP(1)	10 iter. SIP(1)	20 iter. SIP(1)	10 iter. SIP(1)	20 iter. SIP(1)
SIP(1)	10 (0.90)	20 (1.80)	10 (1.40)	20 (2.30)	10 (2.50)	20 (5.31)
CG(1)	24 (1.70)	29 (1.90)	23 (2.00)	27 (2.60)	18 (4.41)	20 (5.31)
SPTDMA(1)	– (–.–)	– (–.–)	– (–.–)	– (–.–)	– (–.–)	– (–.–)
CG	4 (0.90)	6 (1.30)	8 (2.50)	10 (3.00)	10 (8.61)	11 (9.51)
CDCG	4 (1.20)	6 (1.60)	6 (2.20)	7 (2.50)	4 (4.71)	4 (4.71)
CLDCG	3 (1.30)	5 (1.80)	4 (2.40)	5 (2.70)	2 (21.20)	2 (20.90)
GCR	4 (0.90)	5 (1.10)	6 (1.90)	8 (2.80)	5 (4.61)	8 (7.11)
CDGCR	4 (1.00)	5 (1.30)	5 (1.90)	6 (2.30)	4 (4.51)	4 (4.51)
CLDGCR	3 (1.20)	4 (1.40)	4 (2.10)	4 (2.10)	2 (21.00)	2 (20.90)
GCR(1r)	6 (1.20)	11 (2.00)	10 (2.90)	16 (4.11)	8 (6.81)	17 (13.40)
CDGCR(1r)	6 (1.40)	11 (2.30)	8 (2.50)	11 (3.30)	5 (5.41)	6 (5.71)
CLDGCR(1r)	3 (1.10)	5 (1.50)	4 (2.00)	6 (2.60)	2 (20.60)	3 (22.00)
GCR(1t)	4 (0.80)	5 (1.00)	6 (1.80)	8 (2.20)	5 (4.61)	9 (7.41)
CDGCR(1t)	4 (0.80)	5 (1.20)	5 (1.80)	6 (2.10)	4 (4.51)	4 (4.41)
CLDGCR(1t)	3 (1.20)	4 (1.40)	4 (2.00)	4 (2.00)	2 (20.80)	2 (20.70)
GCR(3r)	5 (1.00)	7 (1.40)	6 (1.80)	9 (2.60)	6 (4.91)	9 (7.61)
CDGCR(3r)	5 (1.30)	7 (1.60)	6 (2.00)	6 (2.30)	4 (4.51)	4 (4.41)
CLDGCR(3r)	3 (1.20)	4 (1.40)	4 (1.90)	5 (2.40)	2 (21.00)	2 (20.90)
GCR(3t)	4 (0.80)	5 (1.10)	7 (2.20)	8 (2.50)	5 (4.41)	8 (7.11)
CDGCR(3t)	4 (1.10)	5 (1.50)	5 (1.80)	6 (2.00)	4 (4.51)	4 (4.51)
CLDGCR(3t)	3 (1.20)	4 (1.40)	4 (2.10)	4 (2.10)	2 (21.00)	2 (20.90)

Table E.3: Required inner iterations for solving the pressure-correction system and wall-clock time in parentheses ($\cdot 10^{-1}$ sec.), to obtain the same absolute termination criterion as the SIP(1) method has after 10 and 20 iterations. Testcase I is considered.

	1×1		2×2		4×4	
Method	crit. 1 iter.	crit. 30 iter.	crit. 1 iter.	crit. 30 iter.	crit. 1 iter.	crit. 30 iter.
SIP(1)	0.34	$1.10 \cdot 10^{-1}$	0.39	$2.00 \cdot 10^{-1}$	0.47	$2.60 \cdot 10^{-1}$
SPTDMA(1)	0.51	$2.00 \cdot 10^{-1}$	0.52	$2.10 \cdot 10^{-1}$	0.53	$2.30 \cdot 10^{-1}$
CG(1)	0.87	1.02	0.87	1.03	0.87	1.03
CG	0.95	$1.96 \cdot 10^{-4}$	0.92	$2.56 \cdot 10^{-3}$	0.91	$3.51 \cdot 10^{-2}$
CDCG	0.93	$1.95 \cdot 10^{-4}$	1.01	$9.52 \cdot 10^{-4}$	1.07	$2.75 \cdot 10^{-4}$
CLDCG	0.21	$1.24 \cdot 10^{-2}$	0.23	$9.91 \cdot 10^{-3}$	0.25	$5.03 \cdot 10^{-3}$
GCR	0.34	$2.12 \cdot 10^{-5}$	0.38	$1.71 \cdot 10^{-4}$	0.47	$4.95 \cdot 10^{-4}$
CDGCR	0.34	$2.12 \cdot 10^{-5}$	0.59	$2.53 \cdot 10^{-5}$	0.71	$1.58 \cdot 10^{-6}$
CLDGCR	0.21	$1.13 \cdot 10^{-6}$	0.23	$2.32 \cdot 10^{-7}$	0.25	$7.65 \cdot 10^{-9}$
GCR(1r)	0.34	$7.92 \cdot 10^{-2}$	0.39	$7.26 \cdot 10^{-2}$	0.47	$1.20 \cdot 10^{-1}$
CDGCR(1r)	0.34	$7.92 \cdot 10^{-2}$	0.59	$9.96 \cdot 10^{-3}$	0.71	$5.62 \cdot 10^{-4}$
CLDGCR(1r)	0.21	$1.50 \cdot 10^{-3}$	0.23	$4.49 \cdot 10^{-5}$	0.24	$7.42 \cdot 10^{-7}$
GCR(1t)	0.34	$6.92 \cdot 10^{-5}$	0.39	$1.46 \cdot 10^{-2}$	0.47	$1.78 \cdot 10^{-2}$
CDGCR(1t)	0.34	$6.92 \cdot 10^{-5}$	0.59	$1.23 \cdot 10^{-4}$	0.71	$8.78 \cdot 10^{-6}$
CLDGCR(1t)	0.21	$9.28 \cdot 10^{-6}$	0.23	$5.20 \cdot 10^{-7}$	0.24	$1.49 \cdot 10^{-8}$
GCR(3r)	0.34	$2.26 \cdot 10^{-2}$	0.40	$1.50 \cdot 10^{-2}$	0.47	$7.90 \cdot 10^{-2}$
CDGCR(3r)	0.34	$2.26 \cdot 10^{-2}$	0.59	$2.35 \cdot 10^{-4}$	0.71	$4.75 \cdot 10^{-4}$
CLDGCR(3r)	0.21	$1.27 \cdot 10^{-4}$	0.23	$4.86 \cdot 10^{-6}$	0.24	$2.90 \cdot 10^{-7}$
GCR(3t)	0.34	$4.45 \cdot 10^{-5}$	0.39	$3.36 \cdot 10^{-3}$	0.47	$2.81 \cdot 10^{-3}$
CDGCR(3t)	0.34	$4.45 \cdot 10^{-5}$	0.59	$5.69 \cdot 10^{-5}$	0.71	$2.62 \cdot 10^{-6}$
CLDGCR(3t)	0.21	$3.43 \cdot 10^{-6}$	0.23	$3.74 \cdot 10^{-7}$	0.25	$1.19 \cdot 10^{-8}$

Table E.4: Absolute termination criteria of the solution methods after 1 and 30 iterations, for solving the pressure-correction system. Testcase II is considered.

	1×1		2×2		4×4	
Method	10 iter. SIP(1)	20 iter. SIP(1)	10 iter. SIP(1)	20 iter. SIP(1)	10 iter. SIP(1)	20 iter. SIP(1)
SIP(1)	10 (0.90)	20 (1.80)	10 (0.90)	20 (1.90)	10 (1.10)	20 (2.10)
CG(1)	– (–.–)	– (–.–)	– (–.–)	– (–.–)	– (–.–)	– (–.)
SPTDMA(1)	– (–.–)	– (–.–)	12 (1.30)	22 (2.40)	5 (0.70)	10 (1.30)
CG	8 (1.50)	8 (1.50)	13 (2.50)	13 (2.50)	16 (4.51)	16 (4.21)
CDCG	8 (1.70)	8 (1.50)	7 (1.60)	7 (1.70)	4 (1.50)	5 (1.80)
CLDCG	3 (1.10)	3 (1.10)	1 (0.80)	2 (1.10)	1 (1.20)	1 (1.20)
GCR	5 (0.80)	7 (1.40)	6 (1.40)	8 (1.90)	5 (1.50)	7 (2.10)
CDGCR	5 (1.20)	7 (1.70)	6 (1.60)	6 (1.60)	4 (1.30)	4 (1.50)
CLDGCR	2 (0.90)	2 (0.80)	1 (0.80)	2 (1.00)	1 (1.10)	1 (1.10)
GCR(1r)	7 (1.10)	13 (2.10)	7 (1.30)	9 (1.60)	7 (1.90)	10 (2.40)
CDGCR(1r)	7 (1.20)	13 (2.30)	11 (2.10)	11 (2.20)	7 (2.20)	8 (2.30)
CLDGCR(1r)	2 (0.80)	2 (0.90)	1 (0.80)	2 (1.00)	1 (1.10)	1 (1.10)
GCR(1t)	5 (0.90)	7 (1.30)	7 (1.40)	8 (1.60)	7 (2.00)	10 (2.80)
CDGCR(1t)	5 (1.10)	7 (1.50)	6 (1.40)	7 (1.70)	4 (1.30)	4 (1.50)
CLDGCR(1t)	2 (0.80)	2 (0.90)	1 (0.80)	2 (1.00)	1 (1.10)	1 (1.10)
GCR(3r)	5 (0.90)	7 (1.30)	7 (1.40)	9 (1.70)	7 (1.90)	10 (2.80)
CDGCR(3r)	5 (1.10)	7 (1.40)	6 (1.10)	7 (1.50)	4 (1.40)	5 (1.70)
CLDGCR(3r)	2 (0.90)	2 (0.80)	1 (0.80)	2 (1.00)	1 (1.10)	1 (1.10)
GCR(3t)	5 (1.00)	7 (1.30)	6 (1.20)	9 (1.90)	6 (1.80)	9 (2.60)
CDGCR(3t)	5 (1.30)	7 (1.50)	6 (1.50)	6 (1.50)	4 (1.40)	4 (1.50)
CLDGCR(3t)	2 (0.90)	2 (0.80)	1 (0.80)	2 (1.00)	1 (1.10)	1 (1.10)

Table E.5: Required inner iterations for solving the pressure-correction and wall-clock time in parentheses ($\cdot 10^{-1}$ sec.), to obtain the same absolute termination criterion as the SIP(1) method has after 10 and 20 iterations. Testcase II is considered.

Method	crit. 1 iter.	crit. 30 iter.
SIP(1)	0.48	$9.01 \cdot 10^{-3}$
CG(1)	0.98	$2.13 \cdot 10^{-2}$
SPTDMA(1)	0.43	$5.10 \cdot 10^{-1}$
CG	0.91	$3.01 \cdot 10^{-2}$
CDCG	0.54	$4.88 \cdot 10^{-3}$
CLDCG	0.12	$3.38 \cdot 10^{-5}$
GCR	0.48	$2.43 \cdot 10^{-3}$
CDGCR	0.42	$1.36 \cdot 10^{-4}$
CLDGCR	0.10	$3.95 \cdot 10^{-7}$
GCR(1r)	0.48	$7.92 \cdot 10^{-3}$
CDGCR(1r)	0.42	$6.93 \cdot 10^{-3}$
CLDGCR(1r)	0.10	$3.38 \cdot 10^{-4}$
GCR(1t)	0.48	$4.65 \cdot 10^{-3}$
CDGCR(1t)	0.42	$1.07 \cdot 10^{-3}$
CLDGCR(1t)	0.10	$1.40 \cdot 10^{-6}$
GCR(3r)	0.48	$6.26 \cdot 10^{-3}$
CDGCR(3r)	0.42	$2.00 \cdot 10^{-3}$
CLDGCR(3r)	0.10	$4.80 \cdot 10^{-6}$
GCR(3t)	0.48	$3.78 \cdot 10^{-3}$
CDGCR(3t)	0.42	$3.22 \cdot 10^{-4}$
CLDGCR(3t)	0.10	$5.90 \cdot 10^{-7}$

Method	10 iter. SIP(1)	20 iter. SIP(1)
SIP(1)	10 (3.20)	20 (6.11)
CG(1)	14 (3.91)	21 (5.71)
SPTDMA(1)	– (–.–)	– (–.–)
CG	5 (4.61)	– (–.–)
CDCG	4 (4.61)	12 (10.80)
CLDCG	2 (4.41)	4 (6.11)
GCR	5 (4.41)	8 (7.51)
CDGCR	4 (4.41)	7 (7.51)
CLDGCR	2 (4.01)	4 (6.01)
GCR(1r)	8 (6.11)	10 (7.51)
CDGCR(1r)	7 (6.41)	15 (12.20)
CLDGCR(1r)	2 (4.01)	5 (6.51)
GCR(1t)	5 (4.31)	8 (6.51)
CDGCR(1t)	4 (4.21)	7 (6.71)
CLDGCR(1t)	2 (4.01)	4 (5.71)
GCR(3r)	6 (5.01)	10 (8.21)
CDGCR(3r)	4 (4.31)	8 (7.41)
CLDGCR(3r)	2 (4.01)	5 (6.71)
GCR(3t)	5 (4.61)	8 (7.11)
CDGCR(3t)	4 (4.31)	7 (7.61)
CLDGCR(3t)	2 (4.01)	4 (5.91)

Table E.6: Left table: absolute termination criteria of the solution methods after 1 and 30 iterations, for solving the pressure-correction system. Right table: iterations required and wall-clock time in parentheses ($\cdot 10^{-1}$ sec.), to obtain the same criterion as the SIP(1) method has after 10 and 20 iterations. Testcase III is considered.

Nomenclature

Roman symbols

Symbol	*Description*	*Units*
a	sum of coefficients in the PWI method	[—]
A	matrix of the linear system	[—]
b	source term in the DAS	[—]
B	iteration matrix	[—]
B	postconditioning matrix	[—]
c_p	specific heat at constant pressure	[$\mathrm{m^2\ s^{-2}\ K^{-1}}$]
C	linear algebraic operator in DAS arising from PWI	[—]
d	number of space dimensions	[—]
D	linear algebraic operator in DAS for the velocities	[—]
D	diagonal matrix	[—]
e	internal energy per unit mass	[$\mathrm{m^2\ s^{-2}}$]
e	auxiliary variable for denoting grid positions	[—]
e	global truncation error	[—]
E	total energy per unit mass	[$\mathrm{m^2\ s^{-2}}$]
E	matrix in deflation method	[—]
f	body force	[$\mathrm{kg\ m^{-2}\ s^{-2}}$]
f	dimensionless body force	[—]
f	inhomogeneous term in differential equation	[—]
g	gravitational acceleration	[$\mathrm{m\ s^{-2}}$]
g	boundary value	[—]
G	linear algebraic operator in DAS	[—]
h	cell size	[—]
I	identity matrix	[—]
I	index set	[—]
k	iteration number	[—]
l	iteration number	[—]
L	length scale	[m]
L	discrete operator	[—]
L	approximation of the non-linear operator N	[—]
m	block number	[—]
M	preconditioner	[—]
n	unit normal on a surface	[—]
n	dimension system of equations	[—]

nblock	total number of blocks	[—]
N	number	[—]
N	non-linear algebraic operator in DAS	[—]
N	remainder matrix in the splitting	[—]
N	number of algebraic equations	[—]
p	pressure	[kg m^{-1} s^{-2}]
p	dimensionless pressure	[—]
p	mesh Péclet number	[—]
P	projector in deflation method	[—]
q	energy flux	[kg s^{-4}]
q	source term in transport equation	
q	dimensionless source term in transport equation	[—]
q	auxiliary variable	[—]
Q	projector in deflation method	[—]
Q	linearized momentum equations	[—]
r	residual	[—]
R	pressure-correction matrix	[—]
R	trivial extension matrix	[—]
Re	Reynolds number	[—]
s	rate-of-strain	[m s^{-2}]
s	search direction Krylov method	[—]
S	surface	[—]
t	time	[s]
t	dimensionless time	[—]
T	temperature	[K]
u	velocity	[m s^{-1}]
u	dimensionless velocity	[—]
u	unknown	[—]
U	length scale for the velocity	[m s^{-1}]
v	velocity	[m s^{-1}]
v	dimensionless velocity	[—]
v	variable in Krylov method	[—]
W	computing work	[—]
x	coordinate	[m]
x	dimensionless coordinate	[—]
x	auxiliary variable	[—]
y	initial position of a particle	[—]
y	unknown	[—]
Y	matrix in deflation method	[—]
Z	matrix in deflation method	[—]

Greek symbols

Symbol	*Description*	*Units*
α	relaxation parameter	[—]
α	parameter in orthonormalization process	[—]

Γ	effective transport coefficient	
Γ	artificial boundary	[—]
δ	small parameter	[—]
δ	Kronecker delta function	[—]
Δ	operator in the PWI method	[—]
ε	inverse of the Péclet number	[—]
ε	small disturbance	[—]
κ	(effective) 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	dimensionless density	[—]
σ	stress tensor	[kg m^{-1} s^{-2}]
σ	dimensionless stress tensor	[—]
σ	eigenvalue spectrum	[—]
τ	shear-stress tensor	[kg s^{-2}]
ϕ	property of a material	
φ	property of a material	
φ	dimensionless property of a material	[—]
Ω	domain	[—]

Abbreviations

Abbreviation	*Description*
BC	Boundary Condition
BIM	Basic Iterative Method
CG	Conjugate Gradient
CD	Constant Deflation
CDCG	Constant Deflated Conjugate Gradient
CDGCR	Constant Deflated Generalized Conjugate Residual
CLD	Constant Linear Deflation
CLDCG	Constant Linear Deflated Conjugate Gradient
CLDGCR	Constant Linear Deflated Generalized Conjugate Residual
CDS	Central Difference Scheme
CFD	Computational Fluid Dynamics
DCG	Deflated Conjugate Gradient
DGCR	Deflated Generalized Conjugate Residual
DAS	Differential-Algebraic System
DD	Domain Decomposition
FV	Finite Volume
GCR	Generalized Conjugate Residual
LHS	Left-Hand Side
MGS	Modified Gram-Schmidt
MPI	Message Passing Interface
PDE	Partial Differential Equation

PSD	Positive Definite
RCGS	Reorthogonalized Classical Gram-Schmidt
RHS	Right-Hand Side
SIMPLE	Semi-Implicit Method for Pressure-Linked Equations
SIP	Strongly Implicit Procedure
SPD	Symmetric Positive Definite
SPTDMA	Space Tri-Diagonal Matrix Algorithm
UDS	Upwind Difference Scheme

List of Figures

List of Tables

Bibliography

[1] Bird, R.B., *et al.* (2001). *Transport Phenomena*. New York: Wiley, second edition.

[2] Brakkee, E. (1996). *Domain Decomposition for the Incompressible Navier-Stokes Equations*. PhD thesis. Delft: Delft University Press.

[3] Chan, T.F. and T.P. Mathew (1994). 'Domain decomposition algorithms'. *Acta Numerica*, Vol. 1, pp. 61–141.

[4] Ferziger, J.H. and M. Perić (1999). *Computational Methods for Fluid Dynamics*. Heidelberg: Springer, second edition.

[5] Frank, J. and C. Vuik (1999). 'Parallel implementation of a multiblock method with approximate subdomain solution'. *Appl. Numer. Math.*, Vol. 30, pp. 403–423.

[6] Frank, J. and C. Vuik (2001). 'On the construction of deflation-based preconditioners'. *SIAM J. Sci. Comput.*, Vol. 23, pp. 442–462.

[7] Golub, G.H. and C.F. Van Loan (1996). *Matrix Computations*. Baltimore: The John Hopkins University Press, third edition.

[8] Hoffmann, W. (1989). 'Iterative Algorithms for Gram-Schmidt Orthogonalization'. *Computing*, Vol. 41, pp. 335–348.

[9] Kaasschieter, E.F. (1988). 'Preconditioned conjugate gradients for solving singular systems'. *J. Comp. Applied Math.*, North-Holland, pp. 265–275.

[10] Kim, S. and H-C. Lee (2000). 'On Accuracy of Operator Splitting Techniques for Unsteady Navier-Stokes Equations'. *TEAM seminar*, Department of Mathematics, University of Kentucky.

[11] Kumar, V. *et al.* (1994). *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Redwood City: Benjamin Cummings.

[12] Mansfield, L. (1990). 'On the conjugate gradient solution of the Schur complement system obtained from domain decomposition'. *SIAM J. Numer. Anal.*, Vol. 7, No. 6, pp. 1612–1620.

[13] Nabben, R., C. Vuik (2003) 'A comparison of Deflation and Coarse Grid Correction'. *To be published.*

[14] Nicolaides, R.A. (1987). 'Deflation of Conjugate Gradients with Applications to Boundary Value Problems'. *SIAM J. Numer. Anal.*, Vol. 24, No. 2, pp. 355–365.

[15] Padiy, A., O. Axelsson and B. Polman (2000). 'Generalized Augmented Matrix Preconditioning Approach and its Application to Iterative Solution of Ill-Conditioned Algebraic Systems'. *SIAM J. Matrix Anal. Appl.*, Vol. 22, No. 3, pp. 793–818.

[16] Patankar, S.V. (1980). *Numerical Heat Transfer and Fluid Flow*. New York: McGraw-Hill.

[17] Post, L. (1988). *Modelling of flow and combustion in a glass melting furnace*. PhD thesis. Delft: Delft University Press.

[18] Saad, Y. (2000). *Iterative Methods for Sparse Linear Systems*. Boston: PWS Publishing Company, second edition.

[19] Smith, B.F., *et al.* (1996). *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. New York: Cambridge University Press.

[20] Twerda, A. (2000). *Advanced computing methods for complex flow simulation*. PhD thesis. Delft: Delft University Press.

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

[22] Verweij, R.L. (1999). *Parallel Computing for furnace simulations using domain decomposition*. PhD thesis. Delft: Delft University Press.

[23] Vuik, C. (1996). *Voortgezette numerieke lineaire algebra*. Lecture notes (in Dutch) course 'Voortgezette numerieke lineaire algebra', Delft University of Technology.

[24] Vuik, C. and J. Frank (1999). 'A Parallel Implementation of the Block Preconditioned GCR Method'. *Proc. 7th int. conf. HPCN*, Berlin: Springer, Lecture Notes in Computer Science 1593, pp. 1052–1060.

[25] Vuik, C., A. Segal *et al.* (1999). 'An efficient preconditioned CG method for the solution of layered problems with extreme contrasts in the coefficients'. *J. Comp. Phys.*, Vol. 152, pp. 385–403.

[26] Vuik, C., A. Saghir *et al.* (2000). 'The Krylov accelerated SIMPLE(R) method for flow problems in industrial furnaces'. *Int. J. Numer. Meth. Fluids*, Vol. 33, pp. 1027–1040.

[27] Vuik, C. and J. Frank (2000). 'Deflated ICCG methods applied to problems with extreme contrasts in the coefficients'. *Proc. 16th IMACS World Congress*, Lausanne.

[28] Vuik, C. and J. Frank (2001). 'Coarse grid acceleration of a parallel block preconditioner'. *Future Generation Computer Systems*, Vol. 17, pp. 933–940.

[29] Vuik, C., J. Frank and A. Segal (2001). 'A parallel block-preconditioned GCR method for incompressible flow problems'. *Future Generation Computer Systems*, Vol. 18, pp. 31–40.

[30] Vuik, C. (2002). *Iterative solution methods*. Lecture notes PhD course 'Computational Fluid Dynamics II' at the J.M. Burgerscentrum, Delft University of Technology.

[31] Vuik, C. and A. Saghir (2002). *The Krylov accelerated SIMPLE(R) method for incompressible flow*. Report 02-01 of the Department of Applied Mathematical Analysis, ISSN 1389-6520, Delft University of Technology.

[32] Vuik, C., A. Segal, and J.A. Meijerink (1998). 'An efficient preconditioned CG method for the solution of layered problems with extreme contrasts in the coefficients'. Report 98-20 of the Department of Applied Mathematical Analysis, Delft University of Technology.

[33] Vuik, C., A. Segal, *et al.* (2002). 'A comparison of various deflation vectors applied to elliptic problems with discontinuous coefficients'. *Appl. Numer. Math.*, Vol. 41, pp. 219–233.

[34] Vorst, H.A. van der (2003). *Iterative Krylov Methods for Large Linear systems*. Cambridge: Cambridge University Press.

[35] Wesseling, P. (1991). *An Introduction to Multigrid Methods*. Chichester: Wiley.

[36] Wesseling, P. (2001). *Principles of Computational Fluid Dynamics*. Heidelberg: Springer.

Summary

At the Department of Process Physics at TNO TPD, a large CFD simulation package, called X-stream, is being developed for the glass industry to simulate flows in glass furnaces. The involving equations are the incompressible Navier-Stokes equations, the energy equation, and other equations arising from additional physical models related to the process of glass melting. These equations are discretised with the Finite Volume Method on a colocated grid.

Solving the discretised incompressible Navier-Stokes equations is time consuming, because they are non-linear. In X-stream, the SIMPLE method is used to solve the non-linear system of equations. This is an iterative method where the system in each iteration (outer iteration) is split up into linear systems for the pseudo-velocities and the pressure-correction. The linear systems are solved with a domain decomposition (DD) approach (inner iteration). A DD method is an iterative method in which the spatial domain is decomposed into subdomains for which the equations are solved. In X-stream, an additive Schwarz DD method with minimal overlap is used, in combination with inaccurate subdomain solution.

Due to the elliptic nature of the pressure-correction equation, the Schwarz method converges slowly and the rate of convergence depends on the number of subdomains. Therefore, the main goal in the Master's project is to accelerate the Schwarz method in X-stream. First of all, a new technique called deflation is applied to the pressure-correction system to make the convergence independent on the number of subdomains. In this method, eigenvectors are approximated corresponding to the smallest eigenvalues, and these eigenvalues, which slow down the convergence, are removed. In the Master's project, these eigenvectors are approximated by both constant vectors per subdomain and a combination of constant and linear vectors per subdomain. Secondly, the additive Schwarz method to solve the deflated system is accelerated by a GCR and CG Krylov subspace method. The resulting method is called a Deflated Krylov-Schwarz DD method.

In X-stream, both the GCR-Schwarz and CG-Schwarz with deflation are implemented. First of all, the effect these methods have on solving the pressure-correction system is investigated. Concerning convergence behavior, the Deflated GCR-Schwarz method is superior compared to the default Schwarz method in X-stream. Using constant and linear deflation results in a better convergence rate compared to only constant deflation. Concerning computing time, the Deflated Krylov-Schwarz methods are competitive compared to the default Schwarz method. Furthermore, it was shown that deflation results in a convergence behavior which is more or less independent on the number of subdomains. Secondly, the dependency of the outer iterations on the number of inner iterations, for solving the pressure-correction equation with the Deflated GCR-Schwarz method, and the number of so-called SIMPLE stabilization iterations, is investigated. A SIMPLE stabilization iteration (SSI), which is used in X-stream, is an ad-

ditional 'SIMPLE iteration' for solving the pressure. The results show that a large number of outer iterations and computing time can be gained by applying Deflated GCR-Schwarz acceleration, compared to the default Schwarz method in X-stream. Furthermore, the results show that the Deflated GCR-Schwarz method makes the number of outer iterations independent on the number of inner iterations and SSI iterations. This effect is largest for the case of constant and linear deflation. We therefore conclude that the Deflated GCR-Schwarz method introduces a large robustness. Finally, it was also observed that this method results in less oscillations in the outer iteration residuals.

It can be concluded that the main goal in the Master's project is achieved: Deflated Krylov-Schwarz DD acceleration results in superior convergence behavior compared to the default Schwarz DD method in X-stream, and wall-clock time can be gained.