Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Numerical methods for diffusion problems with a large contrast in the coefficients

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

MASTER OF SCIENCE
in
APPLIED MATHEMATICS

by

RENS BREUNISSEN
Delft, The Netherlands
Januari 2022

MSc thesis APPLIED MATHEMATICS

"Numerical methods for diffusion problems with a large contrast in the coefficients"

RENS BREUNISSEN

Delft University of Technology

To be defended publicly on Monday Januari 24, 2022 at 10:00

Thesis committee

Prof. dr. ir C. Vuik	TU Delft, supervisor
Dr. ir. Y.M. Dijkstra	TU Delft
Ir. E.F. van den Boogaard	Alten NL, daily supervisor
Dr. ir. A. Prins	Alten NL

Student number:	4485181
Master programme:	Applied Mathematics
Specialisation:	Computational Science and Engineering
Faculty:	EEMCS

Januari, 2022 Delft

Abstract

Numerical methods for solving problems with a large contrast in the coefficients are investigated in this report. These types of problems typically appear in basin modeling. Specifically, the deflation and restricted additive Schwarz (RAS) methods are compared for their effectiveness in solving this type of problem in combination with the conjugate gradient method, both in terms of iterations and computation time. It is shown that the RAS method converges to the correct solution in a small amount of iterations. However, the deflation method, in combination with another preconditioner, performs better in terms of computation time. An important observation is that the relative residual can only be used as a reliable stopping criterion when the deflation method is used. The methods can be combined into the DRASCG method, which converges in an extremely small number of iterations. In a parallel environment, the speedup of the RASCG method is limited by a load imbalance in the amount of work required for the application of the preconditioner for each subdomain. The deflation method obtains good speedup, that is close to the ideal speedup. The methods are compared when the number of subdomains is increased. It is shown that a classic data distribution is not effective for this type of problem. The deflation method is shown to be robust for a physics based domain decomposition.

Contents

1 Introduction

Basin modeling is used to reproduce the evolution of a sedimentary basin over a period up to hundreds of millions of years. Such models involve a changing domain due to the effects of deposition, erosion and compaction. At every time-step, computations are performed to find values for temperature and pressure within the different rock layers. The driving force behind these models is the change in fluid pressures within the rock layers, caused by the compaction of the rock under pressure from the deposition of new sediments. Knowledge of the history of the fluid pressure is necessary to discover if the conditions were right for the generation of petroleum and the formation of reservoirs where it can accumulate.

Mathematical models include a wide range of calculations, including computation of temperature, rock and fluid pressure, compaction of the rock layers and changing rock properties. An important component is the computation of fluid pressures throughout the domain at every iteration. Often, the different rock layers in the earth's crust have different properties. Specifically, large contrasts between the permeability coefficients of different layers can be large, up to an order of 10^7. Problems with large contrast in the coefficients also arise in other fields, such as structural mechanics for composite materials [1].

Mathematical models for fluid pressure problems are derived from Darcy's law and the conservation of mass. This results in a time-dependent diffusion equation. In this report, a two-dimensional, time independent, layered problem is defined with a large contrast in the coefficients.

The finite element method is applied to discretise the diffusion equations, due to the flexibility it allows in an application with a changing domain. The finite element discretisation leads to a linear system of equations that needs to be solved at every time-step. Practical applications often involve large three-dimensional domains with a large number of elements. This results in a large linear system to be solved. Although the system is sparse, it is well known that direct methods are not feasible for large systems both due to fill-in requiring too much memory to fit in core, and the number of operations becoming infeasible for large problems.

Therefore, iterative methods are used to solve these types of problems. As the linear system is symmetric, a logical candidate is the preconditioned conjugate gradient method (PCG). The rate of convergence of this method is heavily dependent on the distribution of eigenvalues of the matrix. The large contrast in coefficients leads to an ill-conditioned matrix. Standard preconditioning methods such as incomplete Cholesky preconditioning are not sufficient to improve the convergence rate for this type of problem.

Deflation methods in combination with another preconditioner have been used to greatly improve the convergence rate of the PCG method for this type of problem by effectively projecting away the small eigenvalues associated with the jump in coefficients [2]. In particular, subdomain deflation has been shown to be very appealing in a parallel environment.

Additionally, the restricted additive Schwarz method (RAS) is investigated. As a preconditioner, it is effective in speeding up the convergence of the CG method, and the method is known to be well-suited to parallelisation, as it requires little communication [3]. The method involves the computation of a correction on each subdomain using an iterative method. If the distribution of subdomains is based on the physical layout of the domain, each subdomain can be chosen to contain only elements with similar coefficients. The linear systems associated with the subdomains are therefore much better conditioned than the large system.

In this thesis, the deflation and RAS methods will be investigated for a layered problem with large contrast in the coefficients. They will be compared both in a sequential and a parallel environment. Additionally, it is investigated whether these methods can be effectively combined.

The methods are tested in parallel for a small number of processors. Additionally, a comparison

is made between data based and physics based distribution of the data in a parallel environment.

The solvers and preconditioners used in this report are tested using the PETSc library ([4], [5], [6]), which contains implementations for all the used methods and allows for an easy comparison. For parallel timing, the National Supercomputer Snellius is used. We thank SURF (www.surf.nl) for the support in using the National Supercomputer Snellius.

The outline of this report is as follows: in Section 2, the problem description is outlined. Theoretical background for the methods used in this report is provided in Section 3. In Section 4, the behaviour of the PCG method is explored on a uniform domain. Then, the deflation and RAS methods are analysed for the layered problem in Section 5. In Section 6, parallel results are shown. Finally, in section 7, the main conclusions of this report are summarised.

2 Problem Description

In this section, two model problems are defined. One with a uniform domain, the other with 7 layers with a large contrast in the coefficients. First, a fluid pressure model is derived. This is used as motivation to create the two model problems. Then, the two model problems are defined. Finally, some attention is given to the discretisation of the model problems, as well as the stopping criteria and starting guess used for the linear solvers.

2.1 Fluid pressure model

Mathematical fluid pressure models are derived using Darcy's law and the conservation of mass. A short derivation is provided here.

2.1.1 Fluid pressures

Fluid pressure in a porous medium can be divided in different categories:

- Lithostatic pressure is the pressure caused by the overburden load of the layers above. Both the weight of the solid materials and the fluids is taken into account. If the location is offshore, the weight of the seawater column is taken into account as well.
- Pore pressure is the total pressure in the pore fluids. It is the sum of the hydrostatic pressure and the overpressure.
- Hydrostatic pressure is the pressure caused by the overlaying water column. If the location is offshore, the weight of the seawater column is taken into account as well. The density of water is assumed to be constant, but a distinction is made between the densities of pore water and salt water.
- Overpressure is the difference between hydrostatic pressure and pore pressure, it can be seen as the collection of all pressure in the pore fluid caused by other processes than the weight of the overlaying water column (hydrostatic pressure). Overpressure is caused by various processes, such as fluid expansion processes, cementation and overburden load combined with under-compaction.
- Effective stress is the difference between lithostatic pressure and pore pressure (usually the lithostatic pressure is larger).

The relations between these values can be seen in Figure 1.

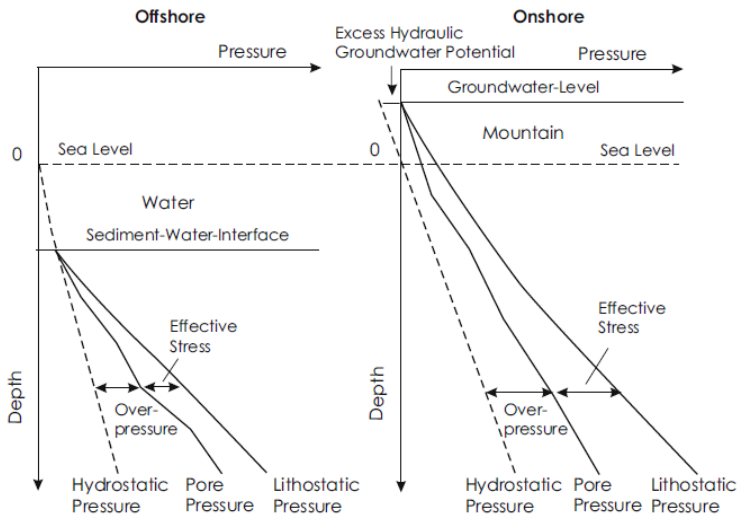


Figure 1: Hydrostatic, pore and lithographic pressure, from [7]

2.1.2 Model derivation

Let v denote fluid discharge velocity. Let u denote the overpressure. Let $\mathbf{k}$ denote the permeability tensor and ν denote the fluid viscosity. Darcy's law describes the flow of a single fluid through a porous medium. It describes a linear relationship between the discharge velocity and the overpressure gradient (assuming that the flow is slow). The permeability tensor is dependent on the rock type.

$$\mathbf{v} = -\frac{\mathbf{k}}{\nu}\nabla u. \tag{1}$$

The overpressure gradient is the driving force behind the flow of water through the porous medium.

Conservation of mass requires that volume discharge from a volume element must be compensated by a change in the contained fluid mass. The fluid mass in a volume element changes when the density changes, or the fluid volume changes. In a porous medium (assuming fully saturated rock), a change in volume is synonymous with a change in porosity. Let ϕ denote the porosity of the rock, and let ρ denote the density of the fluid. This yields:

$$\nabla \cdot \mathbf{v} = -\frac{1}{1-\phi}\frac{\partial \phi}{\partial t} + \frac{1}{\rho}\frac{\partial \rho}{\partial t}. \tag{2}$$

As it is assumed that water is incompressible, the second term vanishes. Changes in porosity can occur as a result of mechanical or chemical compaction. In the basic model, only mechanical compaction is dealt with. Let C denote the compressibility, and let u_l denote the lithostatic potential. This is defined as the difference between the lithostatic pressure and the hydrostatic pressure. The equation for the change in porosity is given by:

$$\frac{\partial \phi}{\partial t} = -C\frac{\partial (u_l - u)}{\partial t}. \tag{3}$$

This provides all the ingredients necessary to derive the overpressure equation. Plugging first equation (1) and then equation (3) into equation (2) yields the following (over)pressure equation:

$$\begin{aligned} -\nabla \cdot \frac{\mathbf{k}}{\nu} \nabla u &= -\frac{1}{1-\phi} \frac{\partial \phi}{\partial t}, \\ &= \frac{C}{1-\phi} \frac{\partial (u_l - u)}{\partial t}, \end{aligned}$$

Rewriting this yields the (over)pressure equation:

$$\frac{C}{1-\phi} \frac{\partial u}{\partial t} - \nabla \cdot \frac{\mathbf{k}}{\nu} \nabla u = \frac{C}{1-\phi} \frac{\partial u_l}{\partial t}. \tag{4}$$

A more detailed derivation of this expression is given by Luo and Vasseur [8].

2.1.3 Boundary conditions

At the surface, the pressure is prescribed. In practical applications this will either be the atmospheric pressure or the sum of the atmospheric pressure and the pressure of the overlaying water column, depending on whether the surface is on land or below water..

It is assumed that the lowest layer is bounded by an impermeable layer. Therefore there is no flux through this boundary. Furthermore, it is assumed that at the vertical boundaries there is no flux, as these boundaries can be taken arbitrarily far away from the basin.

2.2 Model problem 1

The first model problem is defined on a uniform domain. It contains a stationary diffusion problem that is solved on a two-dimensional domain. It is mainly used as a reference for the behaviour of the CG method and preconditioning when compared to the layered problem that will be defined in model problem 2.

Problem
Let $\Omega = [0,1] \times [0,1]$ denote the domain with boundary Γ. Let $\Gamma_D = [0,1] \times \{1\}$ denote the part of the boundary subject to Dirichlet boundary conditions. Let $\Gamma_N = \Gamma \setminus \Gamma_D$ denote the part of the boundary subject to Neumann boundary conditions. Let $u(\mathbf{x})$ denote the pressure for $\mathbf{x} \in \mathbb{R}^2$. Consider the following differential equation:

$$\begin{aligned} -\Delta u &= 0, && \text{in } \Omega, \\ u &= 1, && \text{on } \Gamma_D, \\ \frac{\partial u}{\partial \mathbf{n}} &= 0, && \text{on } \Gamma_N, \end{aligned} \tag{5}$$

where $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ is the 2-dimensional Laplace-operator.

The domain with boundary conditions and an arbitrary number of square elements is shown in Figure 2.

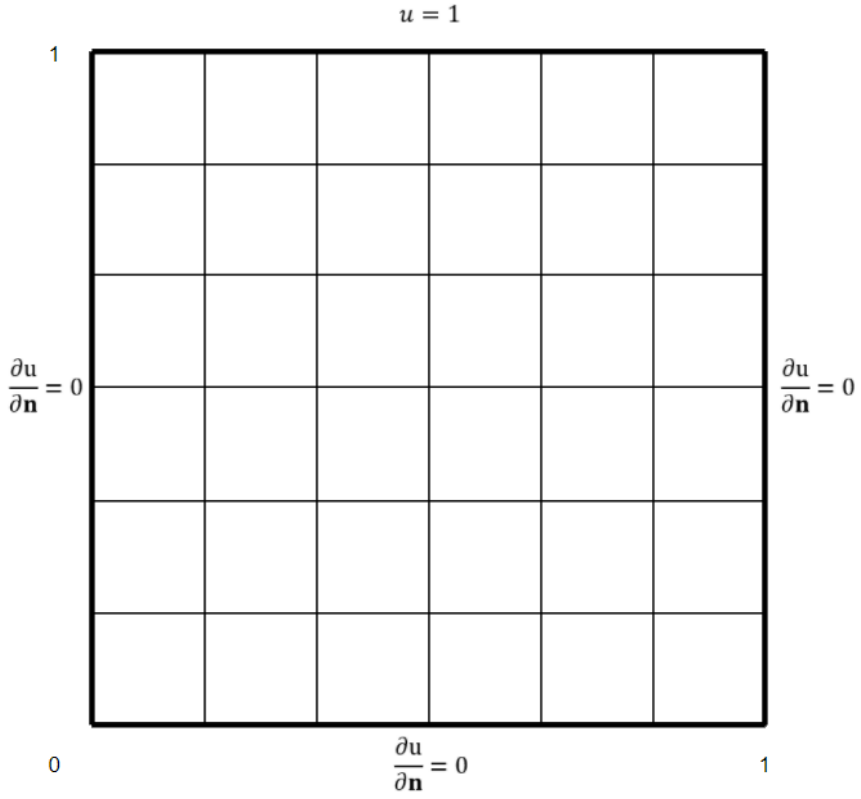


Figure 2: Domain without layers divided into square elements

2.3 Model problem 2

For model problem 2, overpressure equation (4) is simplified further. As the main interest of this research is in the convergence rate for solving layered problems with large contrast in the coefficients, the time-dependent three-dimensional problem is reduced to a stationary two-dimensional problem. The permeability is assumed to be the same in all directions, therefore the tensor in reduced to a one-dimensional variable, and the viscosity of water is assumed to be equal to 1.

Problem
Let $\Omega = [0,1] \times [0,1]$ denote the domain with boundary Γ. Let $\Gamma_D = [0,1] \times \{1\}$ denote the part of the boundary subject to Dirichlet boundary conditions. Let $\Gamma_N = \Gamma \setminus \Gamma_D$ denote the part of the boundary subject to Neumann boundary conditions. Let $u(\mathbf{x})$ denote the pressure for $\mathbf{x} \in \mathbb{R}^2$. Let $\mu(\mathbf{x})$ denote the permeability. Consider the following differential equation:

$$\begin{aligned} -\nabla\left(\mu(\mathbf{x})\nabla\right) u &= 0, && \text{in } \Omega, \\ u &= 1, && \text{on } \Gamma_D, \\ \frac{\partial u}{\partial \mathbf{n}} &= 0, && \text{on } \Gamma_N. \end{aligned} \tag{6}$$

Figure 3: Domain with 7 layers

For this problem the boundary value at $x = 1$ is fixed at 1. This means that the solution to this problem is known, as it is 1 across the entire domain. This is a conscious choice, as it means that it is easy to evaluate the error of the solution, as the exact solution is known.

The domain is composed of 7 layers, with the rock material alternating between sandstone and shale. The domain is shown in figure 13.

The permeability coefficient $\mu(\mathbf{x})$ has different values for the different rock types. Since the main interest is in the contrast between these coefficients, they are defined as follows. Let $\mu_{sandstone}$ denote the permeability of sandstone, and let μ_{shale} denote the permeability of shale. These have the following values:

$$\begin{aligned} \mu_{sandstone} &= 1, \\ \mu_{shale} &= 10^{-7}. \end{aligned}$$

2.4 Discretisation of the model problem

Model equation (13) is discretised using a standard finite element method with quadrilateral elements. The choice for the finite element method is motivated by the application of the methods. In both model problems, the domain is fixed, and a stationary grid is used. Therefore, a finite difference of finite volume method could also be used. However, in a practical application, the domain changes over time due to the deposition of new sediments and the resulting compaction in the underlying rock layers. This inevitably causes deformations in the physical domain, which will have to be accounted for by the discretisation. Such deformations can prove troublesome for the finite difference and finite volume methods. The finite element method offers geometric flexibility, which is why it is the preferred method.

2.4.1 Quadrilateral elements

Quadrilateral elements are used for the discretisation of both model problems. The construction of the element matrices requires computing integrals over all elements. To this end, basis functions are defined on a reference square, and a transformation is used to compute the integrals on an element. For the purposes of this report only square elements are used, but this method works for more general quadrilateral elements.

Let e_{xy} denote a (convex) quadrilateral element in the x, y-plane with corner points $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$. Let $e_{\xi\eta}$ denote the square reference element in the ξ, η-plane defined as $[-1, 1] \times [-1, 1]$. The corner points are $(-1, 1)$, $(1, -1)$, $(1, 1)$ and $(-1, 1)$.

Basis functions
The (bi-)linear basis functions are only known for the reference square. These are defined as follows:

$$\begin{aligned}
\varphi_1(\xi, \eta) &= \frac{1}{4}(1 - \xi)(1 - \eta), \\
\varphi_2(\xi, \eta) &= \frac{1}{4}(1 + \xi)(1 - \eta), \\
\varphi_3(\xi, \eta) &= \frac{1}{4}(1 + \xi)(1 + \eta), \\
\varphi_4(\xi, \eta) &= \frac{1}{4}(1 - \xi)(1 + \eta).
\end{aligned}$$

The transformation from the reference square to an arbitrary quadrilateral element is given by:

$$\begin{aligned}
x &= a_x + b_x\xi + c_x\eta + d_x\xi\eta, \\
y &= a_y + b_y\xi + c_y\eta + d_y\xi\eta.
\end{aligned}$$

The values of the coefficients can be computed explicitly in terms of the corner points of e_{xy}.

Jacobian
The integral of any function $f(x, y)$ over a quadrilateral element e_{xy} can be transformed to an integral over the reference square:

$$\int_{e_{xy}} f(x, y) d\Omega_{xy} = \int_{e_{\xi\eta}} f(\xi, \eta) |det(J)| d\Omega_{\xi\eta},$$

where J is the Jacobian of the transformation defined as

$$J = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix}.$$

The derivatives in the Jacobian can be computed explicitly, this results in the following value for the determinant of the Jacobvian.

$$det(J) = (b_x + d_x\eta)(c_y + d_y\xi) - (b_y + d_y\eta)(c_x + d_x\xi). \tag{7}$$

Rectangular elements
Assume that the quadrilateral element e_{xy} is rectangular. This allows for further simplification of the Jacobian. For a rectangular element, many of the coefficients in the linear transformation to the reference square are 0. This leads to the following transformation:

$$x = a_x + b_x\xi,$$
$$y = a_y + c_y\eta.$$

Let Δx denote the width of the rectangular element, and let Δy denote the height of the rectangular element. Then $b_x = \frac{1}{2}\Delta x$ and $c_y = \frac{1}{2}\Delta y$. This leads to a constant value for the determinant of the Jacobian:

$$det(J) = \frac{1}{4}\Delta x \Delta y.$$

In this case the determinant of the Jacobian is constant and can be taken out of the integral, and the transformation is defined as follows:

$$\int_{e_{xy}} f(x,y)d\Omega_{xy} = \frac{1}{4}\Delta x \Delta y \int_{e_{\xi\eta}} f(\xi,\eta)d\Omega_{\xi\eta}.$$

This transformation is used for the purposes of this report, as square elements are used for the discretisation. For non-rectangular quadratic elements, numerical integration is required, as the determinant of the Jacobian is not constant and can not be taken out of the integral.

2.5 Stopping criteria

Iterative solvers need some sort of stopping criterion to know when the desired accuracy is achieved. Usually, this is done by setting some tolerance for the residual of the found solution, either absolute or relative. Let $\mathbf{r}_0$ denote the initial residual, and let $\mathbf{r}_k$ denote the residual at iteration k. Let ε_{abs} denote the absolute tolerance, and let ε_{rel} denote the relative tolerance. The absolute tolerance is reached when

$$||\mathbf{r}_k||_2 < \varepsilon_{abs}.$$

The relative tolerance is reached when

$$\frac{||\mathbf{r}_k||_2}{||\mathbf{r}_0||_2} < \varepsilon_{rel}.$$

Generally, the relative tolerance can not be smaller than machine precision. For many problems, the relative tolerance is a reliable stopping criterion. It will be shown later in this report that this is not always the case for model problem 2.

In this report, the relative tolerance is used as a stopping criterion and is set at $\varepsilon_{rel} = 10^{-10}$ unless specified otherwise. Relative tolerance is preferred over absolute tolerance as a stopping criterium, as the absolute tolerance does not take the problem size into account.

2.6 Starting guess

To solve the model problem with an iterative method, a starting guess is required. In practical applications, a good starting guess is usually available through the solution at the previous time-step. For both model problems a starting guess with random values in the interval $[0, 1]$ is used. A fixed seed is used for the starting guess, to ensure that the performance of the different methods can be compared accurately.

Another common starting guess is the 0 starting guess. However, for model problem 1 this leads to some unexpected behaviour for certain preconditioners. This unexpected behaviour is caused by the starting guess, and it is not expected that this behaviour will occur in a practical application, therefore this starting guess is not used. This will be explained in more detail in Section 4.4.

2.7 Timing

In this report, timing tests are performed on a personal device for sequential comparison. Parallel timing tests are performed on the Snellius supercomputer. Effort is made to create an isolated environment during the timing tests. On Snellius, this is done by reserving an entire node when running a program, even when not all processors on the node are used. This ensures that the performance is not influenced by other processes running on the node. On the personal device, timing tests are performed shortly after startup, with no other intensive programs in use at the same time. However, these tests are the most error prone, as it is hard to create an isolated environment on a device intended for consumer use.

All timing results shown in this report are taken as the average over 5 runs with the exact same settings. This helps to reduce the influence of outliers in the data.

3 Methods

In this section several methods used in this report are discussed. First, the (preconditioned) conjugate gradient method is presented. After, the Jacobi and incomplete Choleski preconditioners are defined. Then, the deflation method is shown. Finally, the restricted additive Schwarz method is explained.

3.1 The Conjugate Gradient method

The Conjugate Gradient (CG) method is perhaps the most well known Krylov subspace method for solving linear systems of the form:

$$A\mathbf{u} = \mathbf{f}.$$

In this section the CG method is stated simply, along with some of its properties. For a detailed (and intuitive) derivation, see [9].

For the CG method to work, it is essential that the following conditions hold:

- A is symmetric.
- A is positive definite, i.e., for every vector $\mathbf{y} \neq 0$ it holds that $\mathbf{y}^T A\mathbf{y} > 0$.

If these requirements are met, A is called symmetric positive definite (SPD). To fully explain the idea behind the CG method, the A-inner product and A-norm must be defined. The A-inner product for two vectors $\mathbf{y}$ and $\mathbf{z}$ is defined as $(\mathbf{y}, \mathbf{z})_A = \mathbf{y}^T A\mathbf{z}$. The A-norm is defined as $||\mathbf{y}||_A = \sqrt{(\mathbf{y}, \mathbf{y})_A} = \sqrt{\mathbf{y}^T A\mathbf{y}}$. The main idea of the CG method is to compute the iterate $\mathbf{u}_k$ such that it minimises the A-norm of the error over the Krylov subspace of dimension k. That is, if $\mathbf{u}$ is the exact solution, the iterate $\mathbf{u}^k$ computed by conjugate gradient satisfies:

$$||\mathbf{u} - \mathbf{u}^k||_A = \min_{\mathbf{y} \in K^k(A;\mathbf{r}^0)} ||\mathbf{u} - \mathbf{y}||_A. \tag{8}$$

It is not possible to perform such a minimisation for the 2-norm of the error, as it requires knowing the exact value of $\mathbf{u}$.

The CG method can be summarised as follows:

- An initial guess $\mathbf{u}_0$ is made. The initial residual $\mathbf{r}_0 = \mathbf{f} - A\mathbf{u}_0$ is computed.
- In every iteration $k = 1, 2, \ldots$ two scalar values α_k and β_k are computed, as well as the search direction vector $\mathbf{p}^k$. β_k is used to compute search direction vector $\mathbf{p}^k$, α_k is computed such that equation (8) is minimised. For the exact formulas, see Algorithm 1.
- At the end of every iteration the iterate and residual are updated as:

$$\begin{aligned} \mathbf{u}^{k+1} &= \mathbf{u}^k + \alpha_k \mathbf{p}^k, \\ \mathbf{r}^{k+1} &= \mathbf{r}^k - \alpha_k A\mathbf{p}^k. \end{aligned}$$

- When the solution is deemed sufficiently accurate, usually when a certain tolerance is reached, the process is terminated.

Note that the residual is not computed exactly in every itaration. This is because this computation is relatively expensive, whereas updating it in this manner is relatively cheap, as the matrix-vector product $A\mathbf{p}^k$ is already needed in the computation of α_k. However, due to rounding errors the computed residual is not always completely accurate, as it makes no direct comparison between the iterate and $\mathbf{f}$. It is possible to recompute the exact residual when tolerance is reached, and restart if the solution is not sufficiently accurate. The CG algorithm is shown in Algorithm 1, where θ denotes the tolerance and k_{max} denotes the maximum number of iterations. These values are chosen by the user.

Algorithm 1: Conjugate Gradient algorithm

Initialization: $\mathbf{u}^0 = 0$; $\mathbf{r}^0 = \mathbf{f}$; $k = 1$;
while $k < k_{max}$ *and* $(\mathbf{r}^k)^T\mathbf{r}^k > \theta$ **do**
  if $k = 1$ **then**
    $\mathbf{p}^1 = \mathbf{r}^0$;
  else
    $\beta_k = \frac{(\mathbf{r}^{k-1})^T\mathbf{r}^{k-1}}{(\mathbf{r}^{k-2})^T\mathbf{r}^{k-2}}$;
    $\mathbf{p}^k = \mathbf{r}^{k-1} + \beta_k\mathbf{p}^{k-1}$;
  end if
  $\alpha_k = \frac{(\mathbf{r}^{k-1})^T\mathbf{r}^{k-1}}{(\mathbf{p}^k)^T A\mathbf{p}^k}$;
  $\mathbf{u}^k = \mathbf{u}^{k-1} + \alpha_k\mathbf{p}^k$;
  $\mathbf{r}^k = \mathbf{r}^{k-1} - \alpha_k A\mathbf{p}^k$;
  $k = k + 1$;
end while

The Conjugate Gradient method has the nice property that the residuals are orthogonal, and the search vectors are A-orthogonal (i.e. $(\mathbf{p}^i)^T A\mathbf{p}^j = 0$ for $i \neq j$). An interesting effect of this is that conjugate gradient is a finite method, as for $\mathbf{u} \in \mathbb{R}^n$, the Krylov space is equal to $\mathbb{R}^n$ after n iterations. However, for large values of n it is not feasible to perform n iterations, so in practice this property is never used. As discussed before, the result is generally deemed sufficiently accurate when the residual is below a certain threshold.

The convergence of the CG method is tightly linked with the condition number $\kappa_2(A)$ of A. For an SPD matrix A, the condition number is defined as $\kappa_2(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)}$. $\lambda_{max}(A)$ and $\lambda_{min}(A)$ denote the largest and smallest eigenvalues of A respectively (note that all eigenvalues of A are positive, as it is SPD). For the rate of convergence the iterates $\mathbf{u}^k$ satisfy the following inequality (see [10] for a derivation):

$$||\mathbf{u} - \mathbf{u}^k||_A \leq 2\left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right)^k ||\mathbf{u} - \mathbf{u}^0||_A. \tag{9}$$

It is clear from this inequality that for fast convergence (i.e. few iterations) the condition number should be small. Ideally, the eigenvalues of A should be clustered tightly together. This motivates the use of preconditioning to improve the Conjugate Gradient method.

3.2 The Preconditioned Conjugate Gradient method

The system $A\mathbf{u} = \mathbf{f}$ can be preconditioned by multiplying with the inverse of a matrix M: $M^{-1}A\mathbf{u} = M^{-1}\mathbf{f}$. M is called the preconditioner. It is desirable that M approximates A, but

is easier to invert. Furthermore, it is necessary that M is SPD. The main goal of preconditioning is that the eigenvalues of $M^{-1}A$ will be clustered closer together than those of A. Therefore it will require fewer iterations to solve the preconditioned system than the original problem (remember equation (9)).

To use conjugate gradient on the preconditioned system, it is necessary that $M^{-1}A$ is SPD. However, this is generally not the case, even when M is SPD. Therefore, a slightly different approach is needed for preconditioning the CG method.

For Preconditioned Conjugate Gradient, it is required that M is SPD, as for an SPD matrix there always exists a matrix P such that $M = PP^T$. The preconditioned system is $\tilde{A}\tilde{\mathbf{u}} = \tilde{\mathbf{f}}$, where $\tilde{A} = P^{-1}AP^{-T}$, $\tilde{\mathbf{u}} = P^T\mathbf{u}$ and $\tilde{\mathbf{f}} = P^{-1}\mathbf{f}$. The matrix $P^{-1}AP^{-T}$ has the same eigenvalues as $M^{-1}A$.

The algorithm can be rewritten such that it does not use P, which is very useful since it removes the need to compute the decomposition for M. The algorithm is shown in Algorithm 2.

Algorithm 2: Preconditioned Conjugate Gradient algorithm

Initialization: $\mathbf{u}^0 = 0$; $\mathbf{r}^0 = \mathbf{f}$; $k = 1$;
while $k < k_{max}$ *and* $(\mathbf{r}^k)^T\mathbf{r}^k > \theta$ **do**
  $\mathbf{z}^{k-1} = M^{-1}\mathbf{r}^{k-1}$;
  if $k = 1$ **then**
    $\mathbf{p}^1 = \mathbf{z}^0$;
  else
    $\beta_k = \frac{(\mathbf{r}^{k-1})^T\mathbf{z}^{k-1}}{(\mathbf{r}^{k-2})^T\mathbf{z}^{k-2}}$;
    $\mathbf{p}^k = \mathbf{z}^{k-1} + \beta_k\mathbf{p}^{k-1}$;
  end if
  $\alpha_k = \frac{(\mathbf{r}^{k-1})^T\mathbf{z}^{k-1}}{(\mathbf{p}^k)^TA\mathbf{p}^k}$;
  $\mathbf{u}^k = \mathbf{u}^{k-1} + \alpha_k\mathbf{p}^k$;
  $\mathbf{r}^k = \mathbf{r}^{k-1} - \alpha_kA\mathbf{p}^k$;
  $k = k + 1$;
end while

3.2.1 The Jacobi preconditioner

The Jacobi preconditioner is also known as the diagonal scaling preconditioner. The Jacobi preconditioner M_{jac} is defined the matrix containing only the diagonal elements of A.

$$M_{jac} = diag(A).$$

Setting up the preconditioner is computationally cheap, as inverting M_{jac} is reduced to computing the multiplicative inverse of each diagonal element. Application of the Jacobi preconditioner in the PCG method is also computationally cheap, as only 1 multiplication is required for each row in the matrix-vector product.

The Jacobi preconditioner is a very simple preconditioner and therefore its effectiveness in improving the convergence of the CG method is limited for most problems.

3.2.2 The incomplete Cholesky preconditioner

The incomplete Cholesky preconditioner is based of the Cholesky decomposition of matrix A. The Cholesky decomposition is an efficient lower-upper decomposition that can be used by direct

methods. It involves computing a lower-triangular matrix C such that $A = CC^T$.

The Cholesky decomposition is often not used for a direct method, as many problems, including both model problems, have a sparse band matrix A. The Cholesky decomposition leads to a large amount of fill in, resulting in a dense decomposition. This greatly increases the number of computations, which is inefficient for large linear systems.

The incomplete Cholesky preconditioner is computed by performing the Cholesky decomposition, but only using the non-zero elements that coincide with the sparsity pattern of A. That is, the decomposition is not computed for matrix entries where the corresponding entry in A is equal to 0. This is known as incomplete Cholesky with 0 fill-in, or IC(0). It is possible to allow for some fill-in in the construction of $M_{IC(0)}$, and this generally leads to better convergence rates. However, this results is larger computation costs, as well as larger memory requirements. Since the IC(0) preconditioner is already effective at improving the convergence rate, only incomplete Cholesky preconditioning with 0 fill-in is considered in this report.

3.3 Deflation

In situations where the eigenvalues of the linear system matrix are unfavourable, it is possible that the PCG method and the preconditioned GMRES method become less effective. Such a situation might arise for example when the is a large discrepancy in coefficients in a layered problem. In [2], Vuik, Segal and Meijerink show that in such situations, deflation can be used to remove the effect of extreme eigenvalues and thus improve convergence properties. Deflation was first introduced independently by Nicolaides [11] and by Dostál [12] in combination with the CG method. However, deflation can be used in combination with general Krylov methods. In [13], Frank and Vuik give a clear overview of the deflation method. The explanation below is mainly drawn from this article.

Define the projections P and Q by

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

where $Z \in \mathbb{R}^{n\times m}$ Note that if A is symmetric, $Q = P^T$. We assume that $m << n$ and that Z has rank m. Under this assumption, $A_c = Z^T AZ$ is easily factored and SPD. Next, the following splitting is considered:

$$\mathbf{u} = (I - Q)\mathbf{u} + Q\mathbf{u}.$$

Note that $(I - Q)\mathbf{u} = Z(Z^T AZ)^{-1}Z^T A\mathbf{u} = Z(Z^T AZ)^{-1}Z^T\mathbf{f}$ is easily computed, therefore it is only necessary to compute $Q\mathbf{u}$. It is clear that $AQ = PA$, thus one can solve the deflated system:

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

for $\tilde{\mathbf{u}}$. The solution to linear system $A\mathbf{u} = \mathbf{f}$ can be computed as

$$\mathbf{u} = Z(Z^T AZ)^{-1}Z^T\mathbf{f} + Q\tilde{\mathbf{u}}.$$

Note that $\tilde{\mathbf{u}}$ can contain a component in the null space of A, however, this is not a problem as $Q\tilde{\mathbf{u}}$ is unique.

To illustrate the effect of deflation, consider the case where A is SPD and Z is the invariant subspace corresponding to the m smallest eigenvalues of A. Then it can be shown that PA cancels the corresponding eigenvalues (i.e., they are projected out of the residual) while leaving the rest of the spectrum untouched (see [13], lemma 5.2). In this case, the effective condition number is defined with only the non-zero eigenvalues of A:

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

where $\lambda_1 \leq \cdots \leq \lambda_n$ are the eigenvalues of A. The convergence rate of a Krylov method used to solve the deflated system depends on the effective condition number, rather than the regular condition number.

For the non-symmetric case, this result does not necessarily hold, but numerical experiments show similar convergence results.

An important aspect of the deflation method is the choice of restriction matrix Z. Ideally, to increase the rate of convergence, the columns of Z should be the eigenvectors corresponding to the smallest eigenvalues of A. However, eigenvectors are often dense, leading to an expensive deflation

matrix P. Furthermore, finding the eigenvectors of matrix A is not cheap, and the added cost might negate the benefits of using deflation. In practice, Z is often constructed using (sparse) approximations for eigenvectors.

It is also possible to use subdomain deflation (see [13]). This method is related to domain decomposition and multigrid methods (these are discussed further in Section **??**). In this case, the domain Ω is decomposed in m subdomains Ω_j, $j = 1, \ldots, m$, where it is assumed that the Ω_j are simply connected graphs covering Ω. Define index set $\mathcal{I} = \{i | \mathbf{u}_i \in \Omega\}$. The index sets corresponding to the Ω_j are defined as $\mathcal{I}_j = \{i \in \mathcal{I} | \mathbf{u}_i \in \Omega_j\}$. Then Z is defined by:

$$z_{i,j} = \begin{cases} 1, & i \in \mathcal{I}_j, \\ 0, & i \notin \mathcal{I}_j. \end{cases}$$

Here P can be seen as a course grid correction which makes use of very extreme coursening: each subdomain is reduced to a single cell. This course grid correction can make the convergence nearly constant when the number of subdomains is increased.

As a final remark it is noted that deflation can be combined with preconditioning. The results discussed in this section all hold for a preconditioned system. Usually, the deflation method needs to be combined with another preconditioner, as the method's main strength is in projecting away a small amount of problematic eigenvalues from the solution.

3.4 Restricted additive Schwarz

In this section, the restriced additive Schwarz (RAS) method, introduced by Cai and Sarkis [3] is discussed. The RAS method is a small modification of the additive Schwarz (AS) method. It can be used both as an iterative method and as a preconditioner. The iterative method has undesirable convergence properties, and does not converge for all problems. Therefore, the method is usually used as a preconditioner for a Krylov subspace method.

The method requires the definition of a set of overlapping subdomains and a set of non-overlapping subdomain. Application of the method involves the computation of a correction on every overlapping subdomain, but only applying the correction within the non-overlapping subdomain. The values on the boundary of each subdomain are fixed at the value computed in the previous iteration.

Let Ω denote the domain. Let Ω_i for $i = 1, 2, \ldots, n$ denote an overlapping set of subdomains that spans Ω, i.e. $\bigcup_{i=1}^{n} \Omega_i = \Omega$. Furthermore, let $\tilde{\Omega}_i$ for $i = 1, 2, \ldots, n$ denote a non-overlapping set of subdomains that spans Ω such that $\tilde{\Omega}_i \subset \Omega_i$ for all i. That is, $\bigcup_{i=1}^{n} \tilde{\Omega}_i = \Omega$ and $\tilde{\Omega}_i \cap \tilde{\Omega}_j = \emptyset$ for all i, j.

Let R_i denote the rectangular restriction matrix that returns the coefficients defined in the interior of Ω_i when multiplied by $\mathbf{u}$. That is, $R_i\mathbf{u} = \mathbf{u}_{\Omega_i}$. Let $\tilde{R}_i$ denote the rectangular restriction matrix that returns the coefficients defined in the interior of $\tilde{\Omega}_i$ when multiplied by $\mathbf{u}$. That is, $\tilde{R}_i\mathbf{u} = \mathbf{u}_{\tilde{\Omega}_i}$. Let $A_{\Omega_i} = R_i A R_i^T$ denote the restriction of A to subdomain Ω_j.

The RAS preconditioner is defined as follows:

$$M_{RAS} = \sum_i \tilde{R}_i^T A_{\Omega_i}^{-1} R_i,$$

where $\tilde{R}_i$ is the restriction matrix corresponding to subdomain Ω_i without overlap. Note that in practical applications it is often not feasible to compute $A_{\Omega_i}^{-1}$ directly, as this is computationally costly. An iterative method can be used to compute the corrections. The correction is computed by multiplying the RAS preconditioner M_{RAS} with the residual.

The RAS method cuts communication costs in half compared to the AS method, and is faster both in iterations and CPU time. In [3], the method is introduced and some results are shown, but no convergence theory is provided. An algebraic convergence theory is provided in [14].

In [15], Efstathiou and Gander explain why the RAS method performs better than the AS method, by showing that the RAS method is the discrete interpretation of the continuous alternating Schwarz method using block Jacobi. This helps to explain why it converges faster than the AS method. Furthermore, it is shown that AS does not generally converge in the overlap region, whereas the RAS method does.

The main disadvantage of the RAS method is that it is not symmetric, therefore it is more costly to accelerate it with a Krylov subspace method, as a Krylov method for non-symmetric problems is usually used.

4 Uniform domain

In this section, model problem 1 is solved. The problem is defined in Section 2.2, and consists of a uniform domain. The behaviour of the CG method and the PCG method are investigated, and can later be used as a comparison for the layered domain.

4.1 Finite element discretisation

The domain is divided into square elements and equation (5) is discretised using the finite element method, using the basis functions described in Section 2.4. In this section, the derivation of the element matrix and boundary element vector for model problem 1 is shown. These building blocks will be used in the construction of the linear system.

Define the following spaces:

$$H^1(\Omega) = \{u \in L^2(\Omega) | \nabla \cdot u \in L^2(\Omega)\},$$
$$H_0^1(\Omega) = \{u \in H^1(\Omega) | u|_{\Gamma_D} = 0\},$$

where Γ_D is the part of the boundary subject to Dirichlet boundary conditions. The space $H^1(\Omega)$ is known as the Sobolev space of functions with first weak derivative.

Define a test function $\eta \in H_0^1(\Omega)$. To obtain the weak formulation, equation (5) is multiplied by η and integrated over the entire domain Ω. This results in the following equation:

$$-\int_\Omega \eta \Delta u d\Omega = 0. \tag{10}$$

For sufficiently smooth scalar field c and vector field u, Green's theorem states that

$$\int_\Omega c \nabla u d\Omega = -\int_\Omega \nabla c \cdot u d\Omega + \int_\Gamma cu \cdot \mathbf{n} d\Gamma.$$

Note that $\Delta u = \nabla \cdot \nabla u$. Green's theorem can be applied to weak formulation (10) to reduce the order of the derivatives.

$$\int_\Omega \nabla \eta \nabla u d\Omega - \int_\Gamma \eta \nabla u \cdot \mathbf{n} d\Gamma = 0,$$
$$\int_\Omega \nabla \eta \nabla u d\Omega - \int_\Gamma \eta \frac{\partial u}{\partial \mathbf{n}} d\Gamma = 0.$$

Recall that $\Gamma = \Gamma_D \bigcup \Gamma_N$. By definition, the test function η is zero on Γ_D, because it is an element of $H_0^1(\Omega)$. Recall also that on Γ_N, it holds that $\frac{\partial u}{\partial \mathbf{n}} = 0$. It is clear to see that:

$$\int_\Gamma \eta \frac{\partial u}{\partial \mathbf{n}} d\Gamma = \int_{\Gamma_D} \eta \frac{\partial u}{\partial \mathbf{n}} d\Gamma + \int_{\Gamma_N} \eta \frac{\partial u}{\partial \mathbf{n}} d\Gamma,$$
$$= 0.$$

Applying the boundary conditions the weak formulation becomes:

$$\int_\Omega \nabla \eta \nabla u d\Omega = 0. \tag{11}$$

The domain is divided into a finite number of square elements with a total of n discretisation points, and n basis functions $\varphi_j(\mathbf{x})$, $j = 1, 2, \ldots, n$, are defined as described in Section 2.4. The solution is approximated as a linear combination of basis functions:

$$u(\mathbf{x}) = \sum_{j=1}^{n} u_j \varphi_j(\mathbf{x}).$$

This approximation is substituted into weak formulation (11). This introduces n unknown parameters u_j. To solve this n equations are required. Since η and u are in the same space, the test function η can also be represented as a linear combination of basis functions. As η is an arbitrary test function, it is set to equal φ_i for $i = 1, 2, \ldots, n$ to obtain a system of n equations. It is possible to solve this system for the parameters u_j. Note that the solution of the weak formulation is an element of $H_0^1(\Omega)$. The solution to Galerkin's equations is an element of a finite dimensional subspace of $H_0^1(\Omega)$.

Set $u = \sum_{i=1}^{n} u_j \varphi_j$ in weak formulation (11), and $\eta = \varphi_i$ for $i = 1, 2, \ldots, n$ to obtain a system of n equations:

$$\begin{aligned} \int_\Omega \sum_{j=1}^{n} u_j \nabla\varphi_i \nabla\varphi_j d\Omega = 0, \\ \sum_{j=1}^{n} u_j \int_\Omega \nabla\varphi_i \nabla\varphi_j d\Omega = 0. \end{aligned} \tag{12}$$

This set of equations is known as Galerkin's equations.

4.1.1 Element matrix

Let e_k denote an element from the discretisation of Ω with corner points $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$. The element matrix S^{e_k} is of size 4×4 and has elements

$$S_{ij}^{e_k} = \int_{e_k} \nabla\varphi_i \nabla\varphi_j d\Omega,$$

for $i, j \in \{1, 2, 3, 4\}$. This integral is transformed to an integral over the reference square $e_{\xi\eta}$ as described in Section 2.4. As the elements are all rectangular, the determinant of the Jacobian is constant, and the integral becomes:

$$S_{ij}^{e_k} = \frac{\Delta x \Delta y}{4} \int_{e_{k,\xi\eta}} \nabla\varphi_i \nabla\varphi_j d\Omega_{\xi\eta}.$$

As the basis functions are known, these integrals can be computed explicitly. This yields the following element matrix:

$$S^{e_k} = \frac{\Delta x \Delta y}{4} \begin{bmatrix} \frac{2}{3} & -\frac{1}{6} & -\frac{1}{3} & -\frac{1}{6} \\ -\frac{1}{6} & \frac{2}{3} & -\frac{1}{6} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{6} & \frac{2}{3} & -\frac{1}{6} \\ -\frac{1}{6} & -\frac{1}{3} & -\frac{1}{6} & \frac{2}{3} \end{bmatrix}$$

4.1.2 Boundary elements

Consider an element e_k on the Dirichlet boundary with corner points $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$. Let $\mathbf{x}_3$ and $\mathbf{x}_4$ be the points on the Dirichlet boundary. As the points on the Dirichlet boundary will be removed from the system, a correction needs to be made to the right hand side. This can be done by removing the third and fourth row from the element matrix, and moving the values in the third and fourth columns to the right hand side. This yields the following element vector in the boundary elements (for points $\mathbf{x}_1$ and $\mathbf{x}_2$):

$$\mathbf{f}^{e_k} = \frac{\Delta x \Delta y}{4} \begin{bmatrix} \frac{1}{3}u_3 + \frac{1}{6}u_4 \\ \frac{1}{6}u_3 + \frac{1}{3}u_4 \end{bmatrix}.$$

4.2 Conjugate gradient

Discretisation of equation (5) yields a linear system of the form $A\mathbf{u} = \mathbf{f}$. To solve this linear system the most obvious approach is to compute the inverse of A and left-multiply this at both sides of the equation. There are many known methods to solve a problem in this way, most notably methods that use a lower-upper (LU) factorisation. In these methods a lower triangular matrix L and an upper triangular matrix U are computed such that $A = LU$. A special case of an LU-factorisation is the Cholesky factorisation. In this method a lower triangular matrix C is computed such that $A = CC^T$. This method requires less storage (as only one triangular matrix needs to be stored) and the computational cost of the factorisation process is only half of a regular LU-factorisation.

However, both methods require $\mathcal{O}(n^3)$ operations to compute the factorisation of matrix $A \in \mathbb{R}^{n \times n}$ [10]. It should be noted that this number can be reduced for a band matrix, which is the case here. Regardless, the number of operations required remains large. Recall that n is equal to the number of unknowns. It is clear that the cost of the factorisation will be very large for a large number of unknowns. For example, consider a 2-dimensional domain with m unknowns in both the x- and y-direction. The total number of unknowns is m^2. If the number of unknowns is doubled in both directions, the total number of unknowns is $4m^2$. The number of operations required to compute an LU-factorisation becomes $4^3 = 64$ times as large, while the number of unknowns in each direction only increased by a factor of 2. This effect is even more pronounced in 3-dimensional domains. It should be clear that direct methods are not scalable, i.e., the performance of the methods will be worse for a larger problem. Therefore direct methods are not often used for sparse problems.

Since the linear system yielded from the finite element discretisation is symmetric, the CG method is used to solve the linear system. This method is described in section 3.1. As the CG method has some good optimality properties, it is expected to converge to the correct solution quickly. Numerical results show that the method converges in 380 iterations to the desired relative tolerance of 10^{-10}. The residual plot for the CG method can be seen in Figure 4.

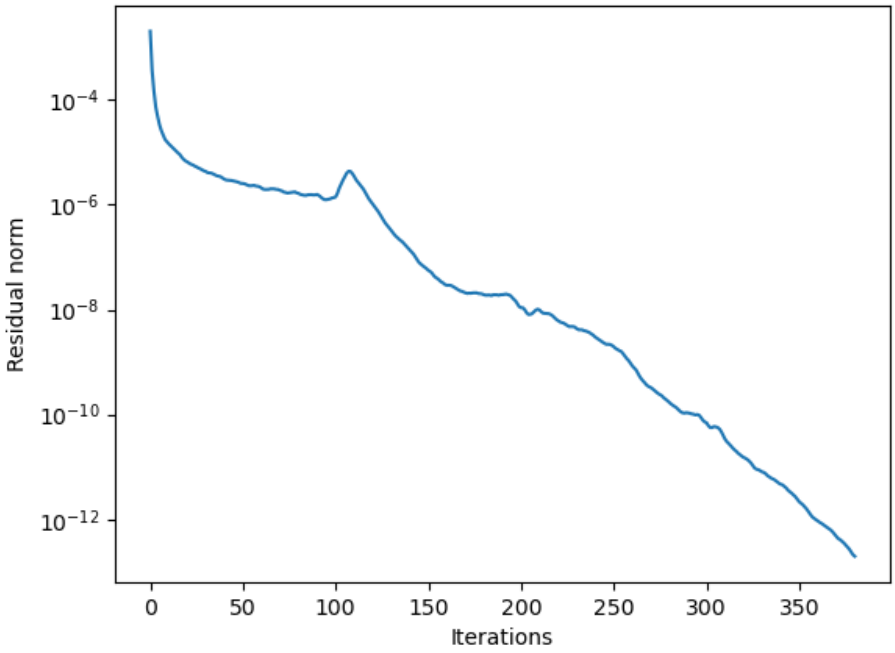
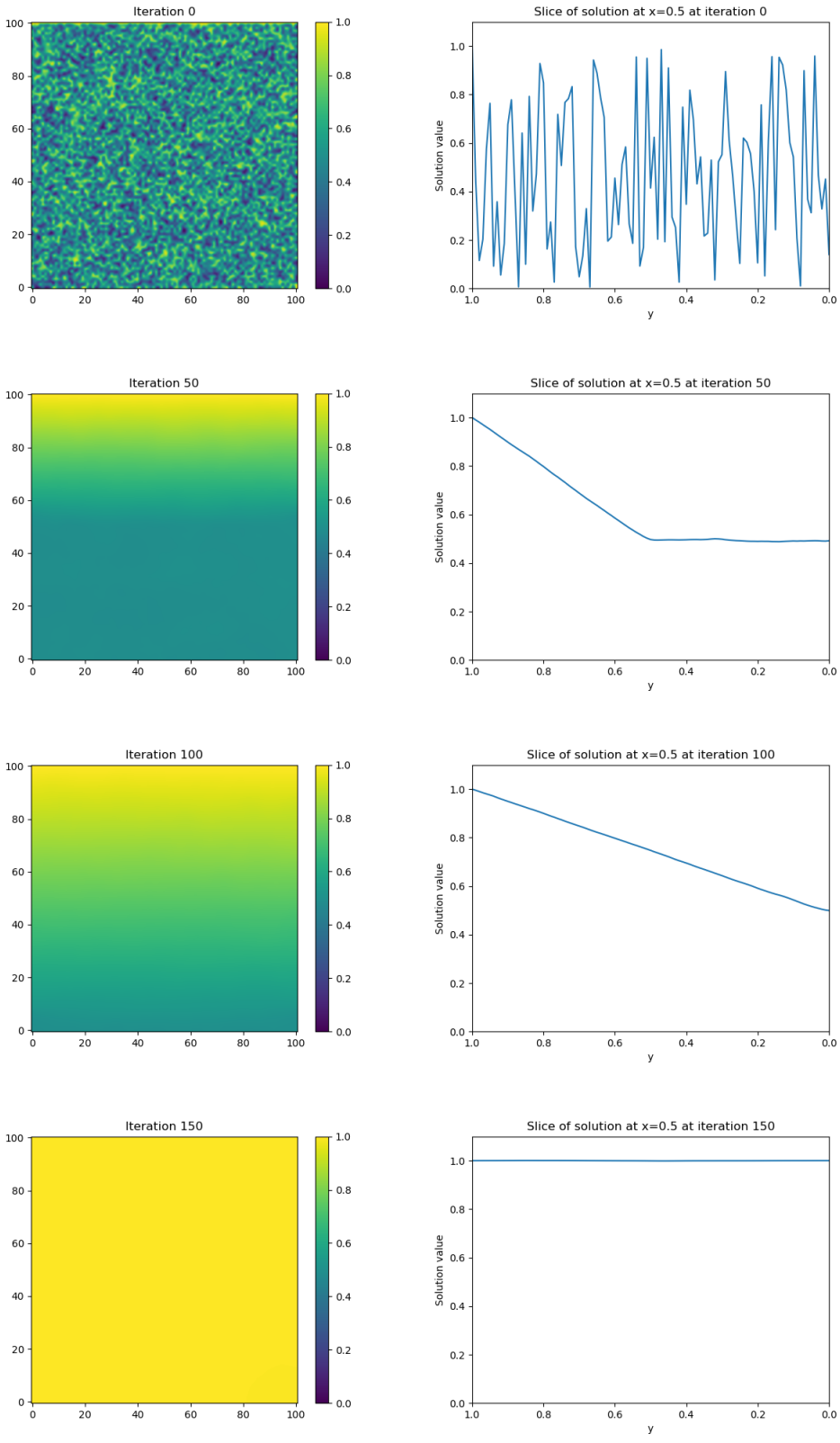


Figure 4: Residual norm plot for the CG method

The evolution of the solution is shown as a function of the number of iterations in Figure 5, both in the entire domain, and a slice of the solution at $x = 0.5$. This gives a good sense of the way the information propagates through the domain. At iteration 0 , the solution takes random values between 0 and 1. This is because the starting guess is taken randomly in the interval $[0, 1]$. The solution is decided by the value on the Dirichlet boundary Γ_D at $y = 1$. This information 'travels' through the domain in the iterative process. At every iteration, each discretisation point only receives information from the discretisation points directly or diagonally adjacent to it. Therefore the information from the Dirichlet boundary needs at least 100 iterations to move through the entire domain. This can be clearly seen in Figure 5. After 50 iterations, the information has travelled halfway through the domain. After 100 iterations, the information has reached the bottom of the domain. In 50 more iterations, at iteration 150, the solution is 1 all across the domain. The rest of the iterations are required to reach the desired relative tolerance of 10^{-10}. This is a tight requirement, which explains why 380 iterations are needed to reach convergence.



(a) Solution in the entire domain (b) Slice of the solution at $x = 0.5$

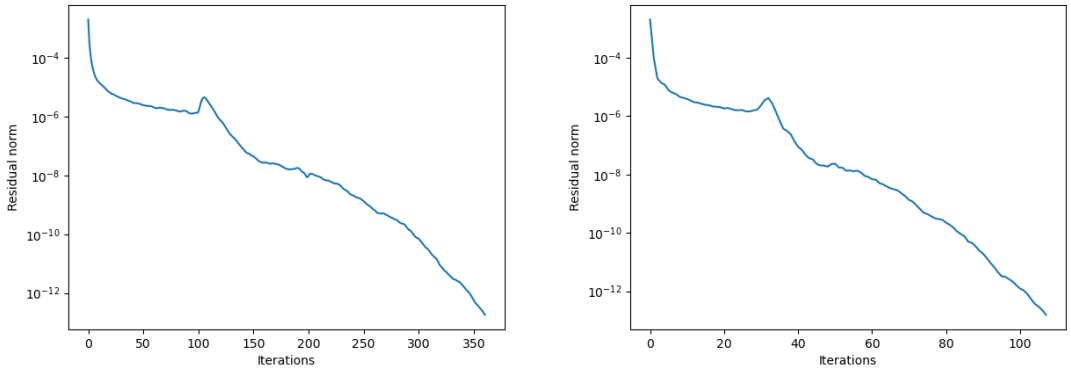
Figure 5: Evolution of the solution using the CG method

4.3 Preconditioned conjugate gradient

The CG method can be accelerated with a preconditioner, which improves the condition number of the linear system. The preconditioned conjugate gradient (PCG) method is described in Section 3.2. Some of the most basic preconditioners are the Jacobi preconditioner and the incomplete Cholesky preconditioner, described in Section 3.2.1 and Section 3.2.2. Here, incomplete Cholesky with zero fill-in (IC(0)) is used. The residual plots for these preconditioners can be seen in Figure 6. These plots look very similar to the residual plot of the CG method without preconditioning.

PCG with Jacobi preconditioning converges in 360 iterations, which is only a very small improvement over the CG method without preconditioning. This is explained by the fact that the diagonal elements of A are all of similar size. In fact, all rows of A corresponding to internal discretisation points (i.e., points that are not on the boundary) have exactly the same value on the diagonal, because all elements are of the same size. Therefore, the Jacobi preconditioner is almost the same as scaling the matrix by a constant, which will not improve the condition number at all. The small improvement of the Jacobi preconditioner can be explained by the discretisation points on the boundary. The rows of A corresponding to these points will have a different value on the diagonal than internal discretisation points.

PCG with IC(0) preconditioning converges in 107 iterations. This is a significant improvement over CG without preconditioning. The IC(0) preconditioner is effectively able to improve the condition number of the linear system.

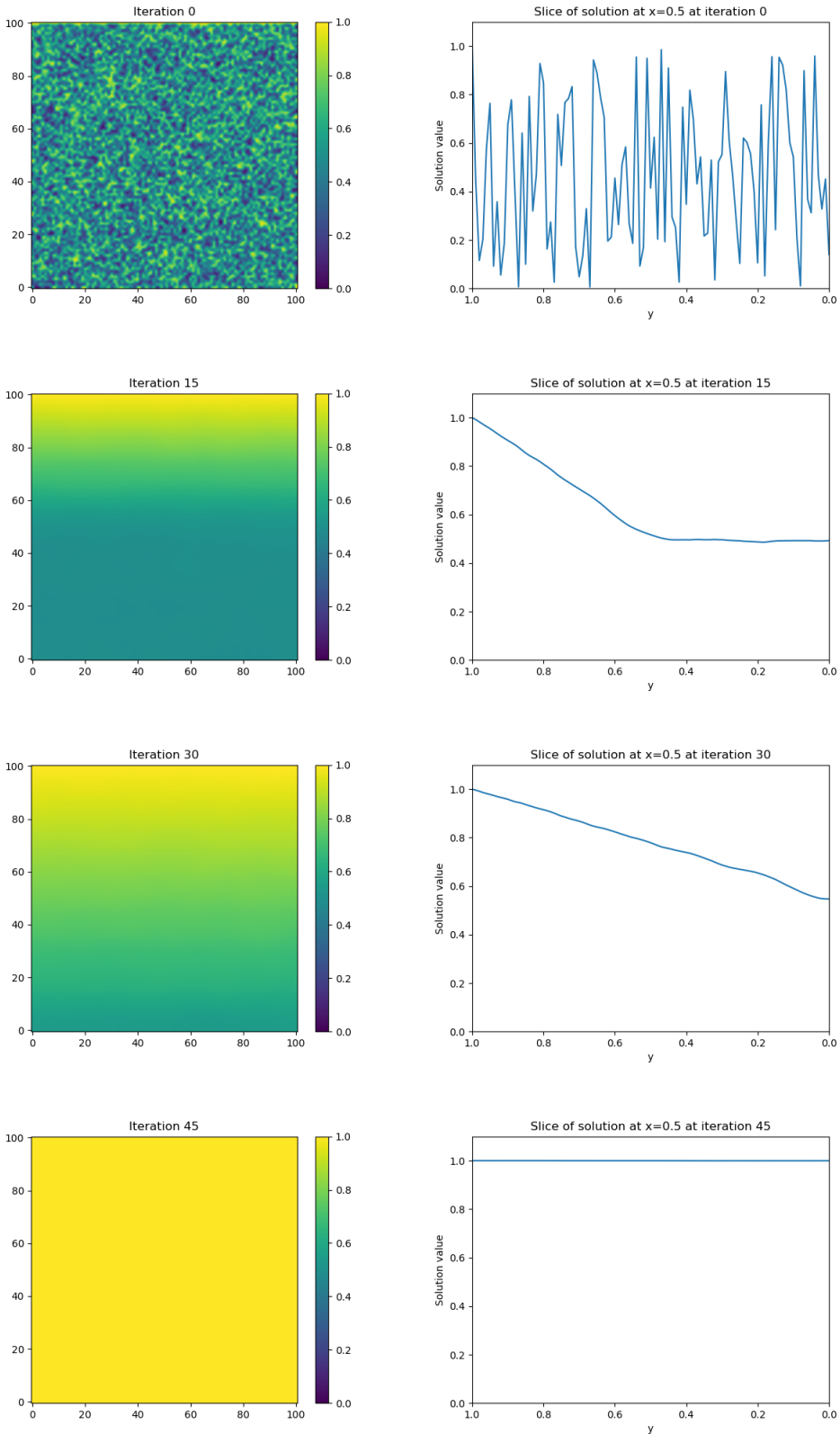


(a) Jacobi preconditioning

(b) IC(0) preconditioning

Figure 6: Residual norm plot for the PCG method

In Figure 7, the evolution of the solution using the PCG method with IC(0) preconditioning is shown. Note that this is very similar to the evolution of the solution for the CG methods without preconditioning. The main difference is the rate of convergence. After 15 iterations, the information has already travelled halfway through the domain. After 30 iterations, the information has travelled through the entire domain. The solution is already good at iteration 45. The rest of the iterations are needed to reach the desired accuracy.



(a) Solution in the entire domain (b) Slice of the solution at $x = 0.5$

Figure 7: Evolution of the solution using the PCG method with IC(0) preconditioning

Finally, in Figure 8 the residual plots for CG without preconditioning, PCG with Jacobi preconditioning and IC(0) preconditioning are shown. Here, PCG with Jacobi preconditioning is denoted as JCG, and PCG with IC(0) preconditioning is denoted as ICCG(0). It is clear that the Jacobi preconditioner does not offer any significant improvement for this problem. The IC(0) preconditioner accelerates the CG method significantly in terms of convergence rate.

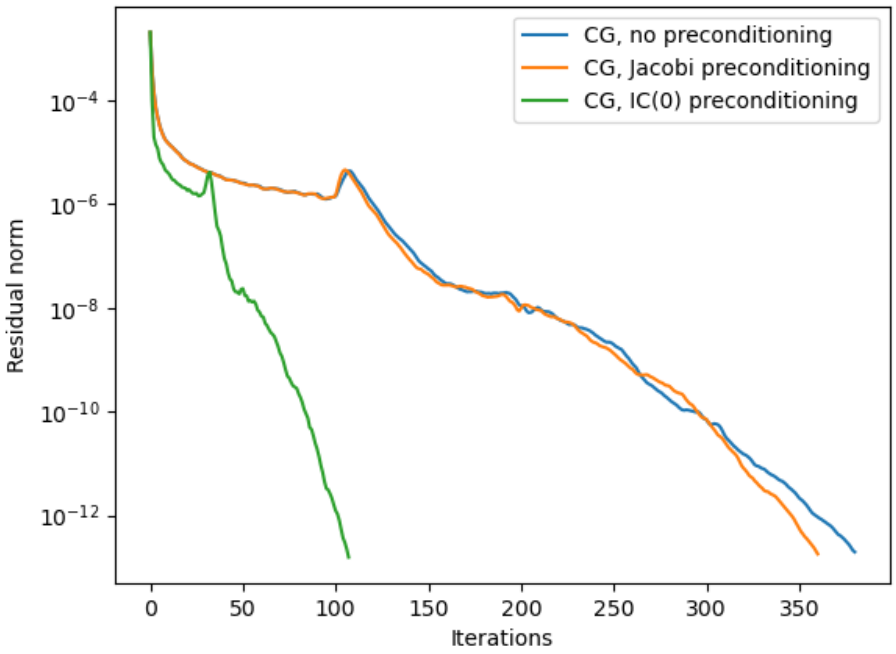


Figure 8: Residual norm plot for the CG, JCG and ICCG(0) methods

In Table 1 the CPU times for the different methods are shown. The CG and JCG methods have similar times, while the ICCG(0) method is roughly twice as fast. Note that the number of iterations required for ICCG(0) is roughly a third of the iterations required for the CG method. However, the speedup is only 2. This is explained by the extra work required in setting up and applying the preconditioner matrix.

Method	**CPU time (s)**	**Speedup compared to CG**
CG	0.237	-
JCG	0.237	1
ICCG(0)	0.122	1.94

Table 1: CPU time for the CG, JCG and ICCG(0) methods

4.4 Choice of starting guess

For all previous results, the starting guess was random, with values in the interval $[0, 1]$. Another logical candidate as a starting guess is the zero starting guess. However, this starting guess gives some weird results for this problem. In Figure 9, the residual plots for the CG, JCG and ICCG(0) methods are shown. The plots for the CG and ICCG(0) methods look the same as before, and require 296 and 110 iterations to reach convergence respectively. The JCG method, however, suddenly reaches convergence after 100 iterations. This is unexpected behaviour, and it is odd that the solution becomes that much more accurate in one iteration.

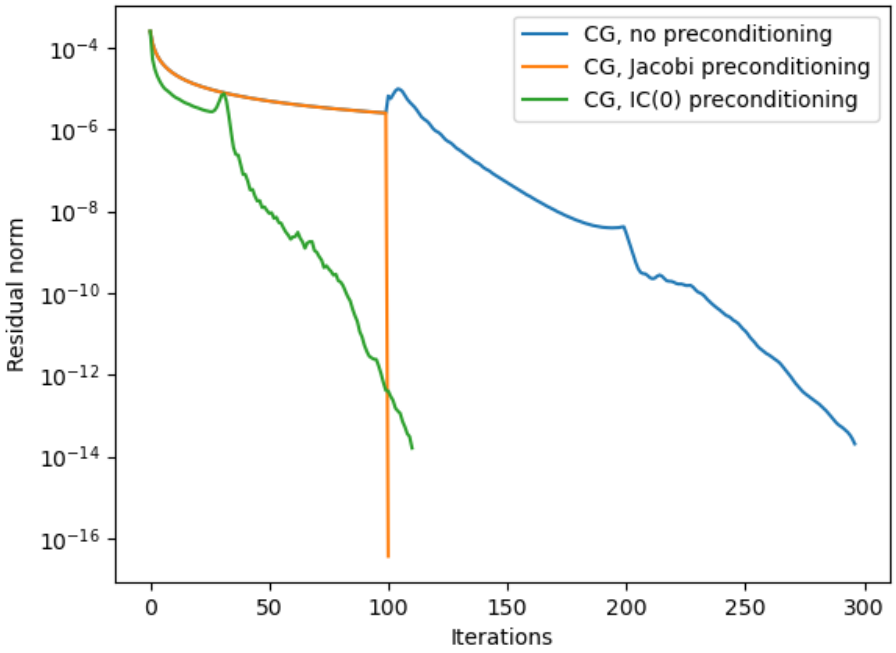
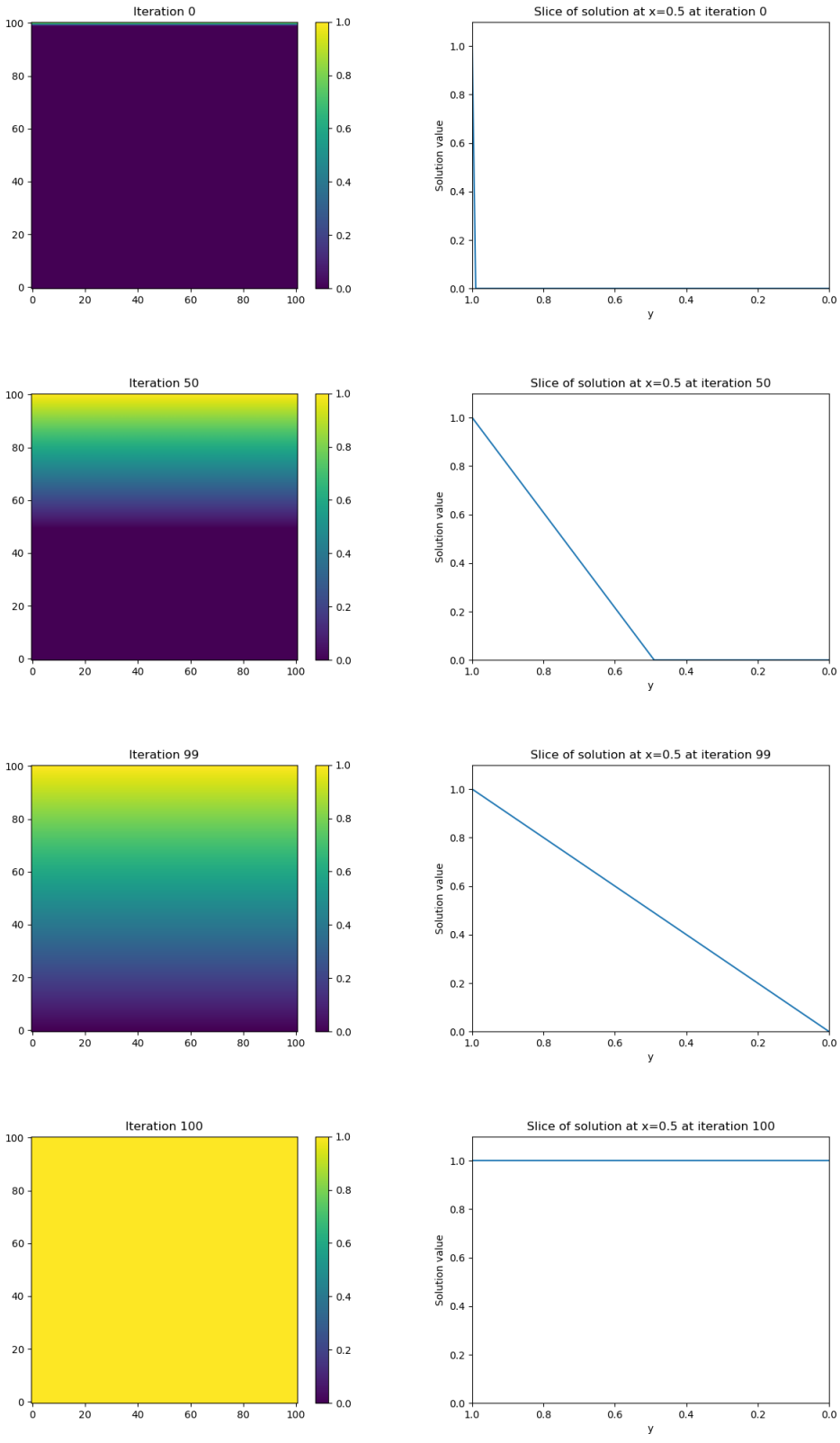


Figure 9: Residual norm plot for the CG, JCG and ICCG(0) methods, using a zero starting guess

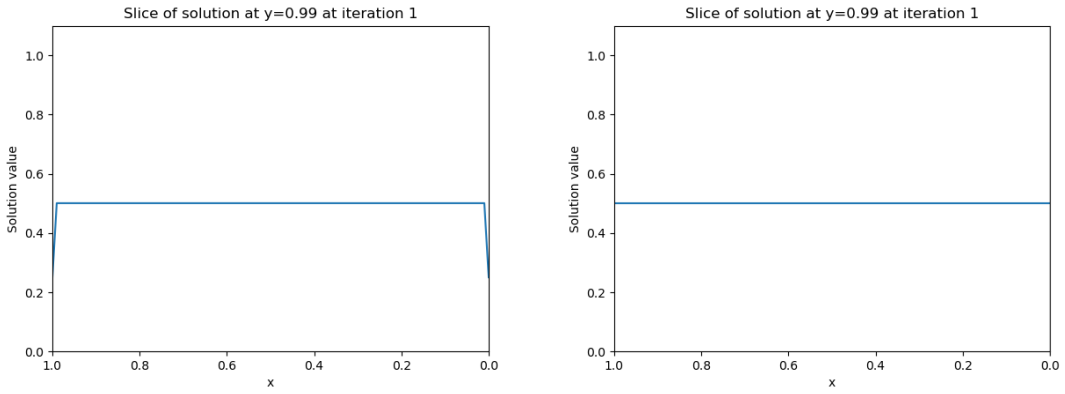
The evolution of the solution for the JCG method can be seen in Figure 10. The behaviour is the same as before for the first 99 iterations, and the information propagates through the domain as expected from the JCG method with random starting guess. However, at iteration 100 the correct solution is found quite suddenly and accurately.



(a) Solution in the entire domain (b) Slice of the solution at $x = 0.5$

Figure 10: Evolution of the solution using the JCG method, using a zero starting guess

The answer to this unexpected behaviour can be found by looking more closely at the solutions at each iteration. In Figure 12, a slice of the solution is shown at $y = 0.99$ (the second row of discretisation points from the top). On the left the solution is shown for the CG method, on the right for the JCG method. With the CG method, the solution is mostly the same, except on the left and right boundaries. With the JCG method, the solution is exactly the same on the entire slice.



(a) CG method

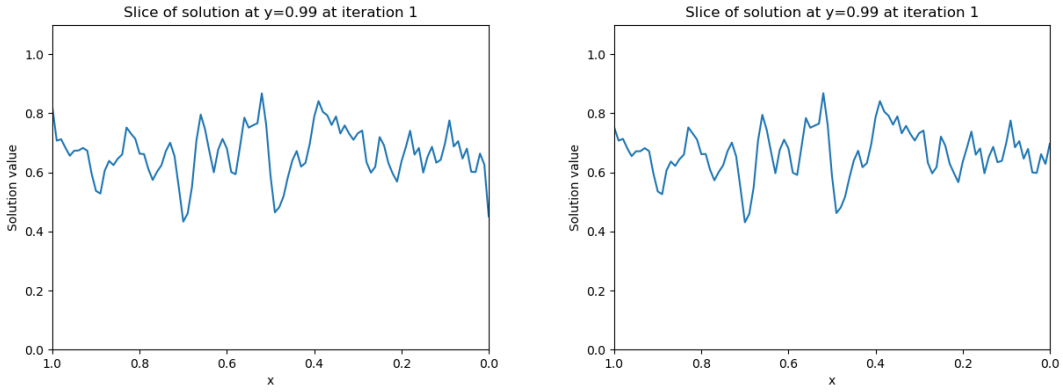
(b) JCG method

Figure 11: Solution value at y=0.99 after one iteration, using a zero starting guess

In fact, this phenomenon occurs for the solution at every iteration, and for every row of discretisation points in the x-direction. The diagonal scaling of the Jacobi preconditioner has removed all difference in the x-direction, which effectively changes the 2-dimensional problem to 101 1-dimensional problems that are all exactly the same.

Recall the following optimality property of the CG method: for solution vector $\mathbf{u} \in \mathbb{R}^n$, the CG method finds the exact solution in n iterations. This property is usually not used much, as often the value of n is very large. In fact, in the given problem the value of n is $100 \cdot 101 = 10100$ (since the discretisation points on Γ_D) are eliminated from the system). However, the number of discretisation points in y-direction is 101, with one of those points on Dirichlet boundary Γ_D. Therefore, there are only 100 unknowns in each column.

Due to the diagonal scaling, the JCG method is effectively solving a 1-dimensional problem with 100 unknowns. Therefore, it is able to find the solution with great accuracy after 100 iterations. In a practical application, it is not expected that the pressures are equal in x-direction during the iterative process. Therefore the behaviour of the JCG method for this starting guess is not representative. This motivates the use of a random starting guess instead.



(a) CG method

(b) JCG method

Figure 12: Solution value at y=0.99 after one iteration, using a random starting guess

5 Layered domain

Model problem 2 is defined in Section 2.3. It consists of a domain Ω on the unit square with 7 layers. For completeness, the equations are repeated here.

Problem
Let $\Omega = [0,1] \times [0,1]$ denote the domain with boundary Γ. Let $\Gamma_D = [0,1] \times \{1\}$ denote the part of the boundary subject to Dirichlet boundary conditions. Let $\Gamma_N = \Gamma \setminus \Gamma_D$ denote the part of the boundary subject to Neumann boundary conditions. Let μ denote the permeability. Consider the following simple differential equation:

$$\begin{aligned} -\nabla \left(\mu \nabla\right) u &= 0, && \text{in } \Omega, \\ u &= 1, && \text{on } \Gamma_D, \\ \frac{\partial u}{\partial \mathbf{n}} &= 0, && \text{on } \Gamma_N. \end{aligned} \tag{13}$$

For this simple problem the boundary value at $x = 1$ is fixed at 1. This means that the solution to this problem is rather simple, as it is 1 across the entire domain. However, this means that it is easy to evaluate the error of the solution, as the exact solution is known.

The domain is composed of 7 layers, with the rock material alternating between sandstone and shale. The domain is shown in figure 13.

Figure 13: Domain with 7 layers

For the rock properties the main interest is in the contrast in coefficients. Therefore the permeability values are taken to be 1 in sandstone and 10^{-7} in shale.

The domain is divided into 100×100 square elements. The interface between two layers coincides exactly with the interface between two rows of elements. In other words, each element is completely contained within one layer, and therefore only has the material properties of one material type. Note that the number of elements does not neatly divide into the number of layers, as dividing 100 by 7 leaves a remainder of 2 elements. For implementation purposes these are added to the bottom two layers, these are therefore slightly thicker than the other layers.

5.1 Finite element discretisation

The domain is divided into square elements and equation (13) is discretised using the finite element method. The derivation of the element matrix and boundary element vector is very similar to the derivation for model problem 1. The main difference is that $\mu(\mathbf{x})$ is no longer constant throughout the domain. Since it is not constant, it can not be taken out of the integral for the weak equation and Galerkin's equations.

Since the derivation of Galerkin's equations is similar to the one shown for test problem one, it is omitted here. For $i = 1, 2, \ldots, n$, the set of Galerkin's equations for model problem 2 is given by:

$$\sum_{j=1}^{n} u_j \int_{\Omega} \mu \nabla \varphi_i \nabla \varphi_j d\Omega = 0. \tag{14}$$

This set of equations is known as Galerkin's equations.

5.1.1 Element matrix

Let e_k denote an element from the discretisation of Ω with corner points $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$. The element matrix S^{e_k} is of size 4×4 and has elements

$$S_{ij}^{e_k} = \int_{e_k} \mu \nabla \varphi_i \nabla \varphi_j d\Omega,$$

for $i, j \in \{1, 2, 3, 4\}$. Note that μ is constant within the element, and can thus be taken out of the integral. Let μ_{e_k} denote the value of μ within element e_k. This integral is transformed to an integral over the reference square $e_{\xi\eta}$ as described in Section 2.4. As the elements are all rectangular, the determinant of the Jacobian is constant, and the integral becomes:

$$S_{ij}^{e_k} = \mu_{e_k} \frac{\Delta x \Delta y}{4} \int_{\Omega_{\xi\eta}} \nabla \varphi_i \nabla \varphi_j d\Omega_{\xi\eta}.$$

As the basis functions are known, these integrals can be computed. This yields the following element matrix:

$$S^{e_k} = \mu_{e_k} \frac{\Delta x \Delta y}{4} \begin{bmatrix} \frac{2}{3} & -\frac{1}{6} & -\frac{1}{3} & -\frac{1}{6} \\ -\frac{1}{6} & \frac{2}{3} & -\frac{1}{6} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{6} & \frac{2}{3} & -\frac{1}{6} \\ -\frac{1}{6} & -\frac{1}{3} & -\frac{1}{6} & \frac{2}{3} \end{bmatrix}$$

5.1.2 Boundary elements

Consider an element e_k on the Dirichlet boundary with corner points $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$. Let $\mathbf{x}_3$ and $\mathbf{x}_4$ be the points on the Dirichlet boundary. As the points on the Dirichlet boundary will be removed from the system, a correction needs to be made to the righthand side. This can be done by removing the third and fourth row from the element matrix, and the moving the values in the third and fourth columns to the right hand side. This yields the following element vector in the boundary elements (for points $\mathbf{x}_1$ and $\mathbf{x}_2$):

$$\mathbf{f}^{e_k} = \mu_{e_k} \frac{\Delta x \Delta y}{4} \begin{bmatrix} \frac{1}{3} u_3 + \frac{1}{6} u_4 \\ \frac{1}{6} u_3 + \frac{1}{3} u_4 \end{bmatrix}.$$

5.2 Conjugate gradient and preconditioning

In Section 4, the CG method was used to solve the problem on a domain with no layers. Additionally, two preconditioning methods were used to accelerate the CG method, namely the Jacobi preconditioner and the IC(0) preconditioner. As the linear system arising from the finite element discretisation is SPD, these methods can also be used to solve this problem. The iterations numbers for these methods can be seen in Table 2.

Method	**#iterations**
CG	9163
JCG	726
ICCG(0)	218

Table 2: Iteration numbers for the CG, JCG and ICCG(0) methods

The number of iterations is much larger for all methods when compared to the problem without layers. Most notably, the CG method without preconditioning needed just 380 iterations to converge for the domain without layers. However, with layers the number of iterations required to reach a solution is 9163, roughly 24 times as many. In Figure 14, the residual plot for the CG method is shown. The residual norm improves rapidly over the first 100 iterations, but afterwards it shows greatly oscillating behaviour, until finally reaching convergence after 9163 iterations.

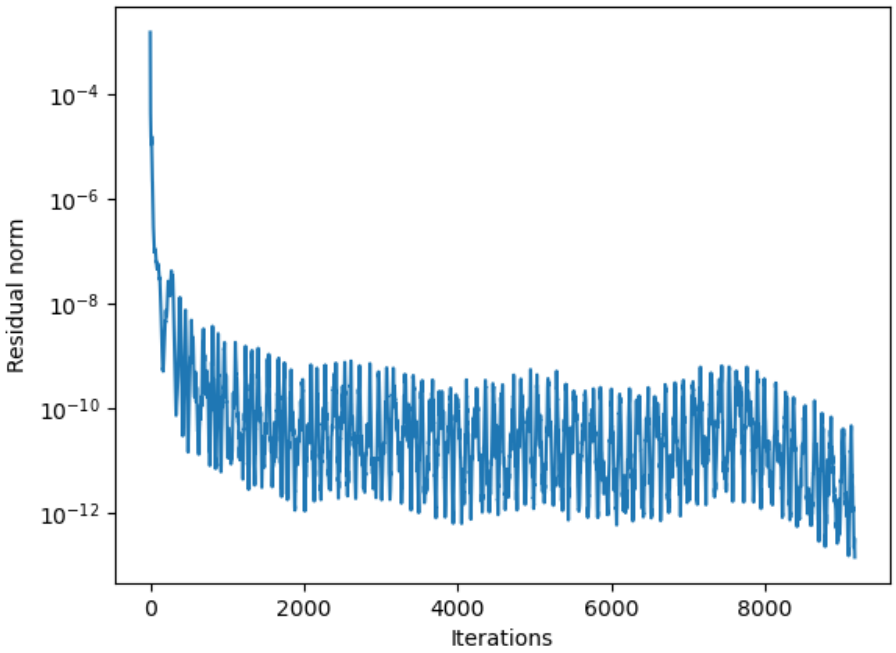
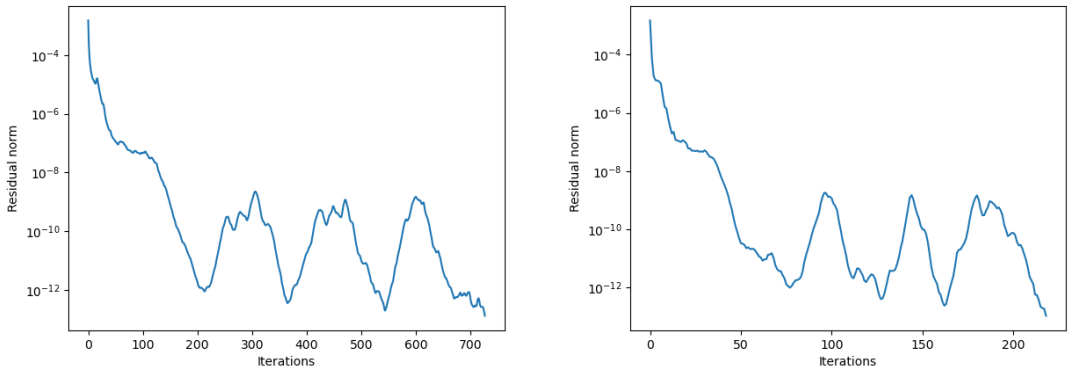


Figure 14: Residual norm plot for the Conjugate Gradient method

In Figure 15, the residual plots are shown for the JCG and the ICCG(0) method. These figures do not show the same oscillating behaviour as the CG method. However, both figures noticably

have three 'bumps' in the residual norm. Every time, the residual decreases significantly, only to increase again after a number of iterations. Finally, after the three of such 'bumps', the methods converge to the desired accuracy.



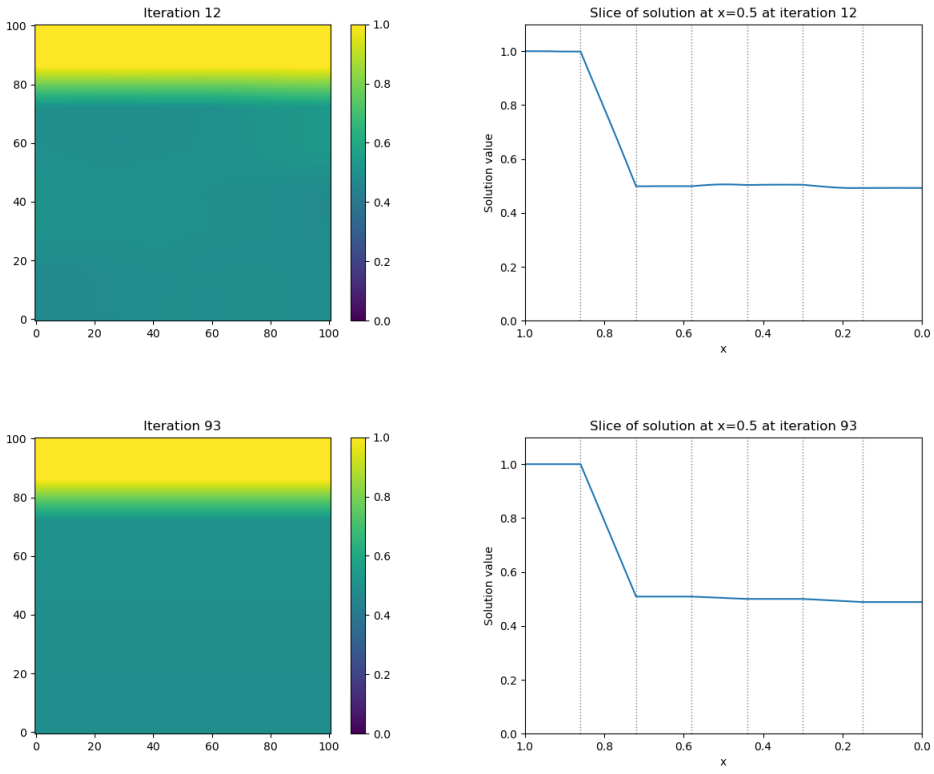
(a) JCG method

(b) ICCG(0) method

Figure 15: Residual norm plots for the JCG and ICCG(0) methods

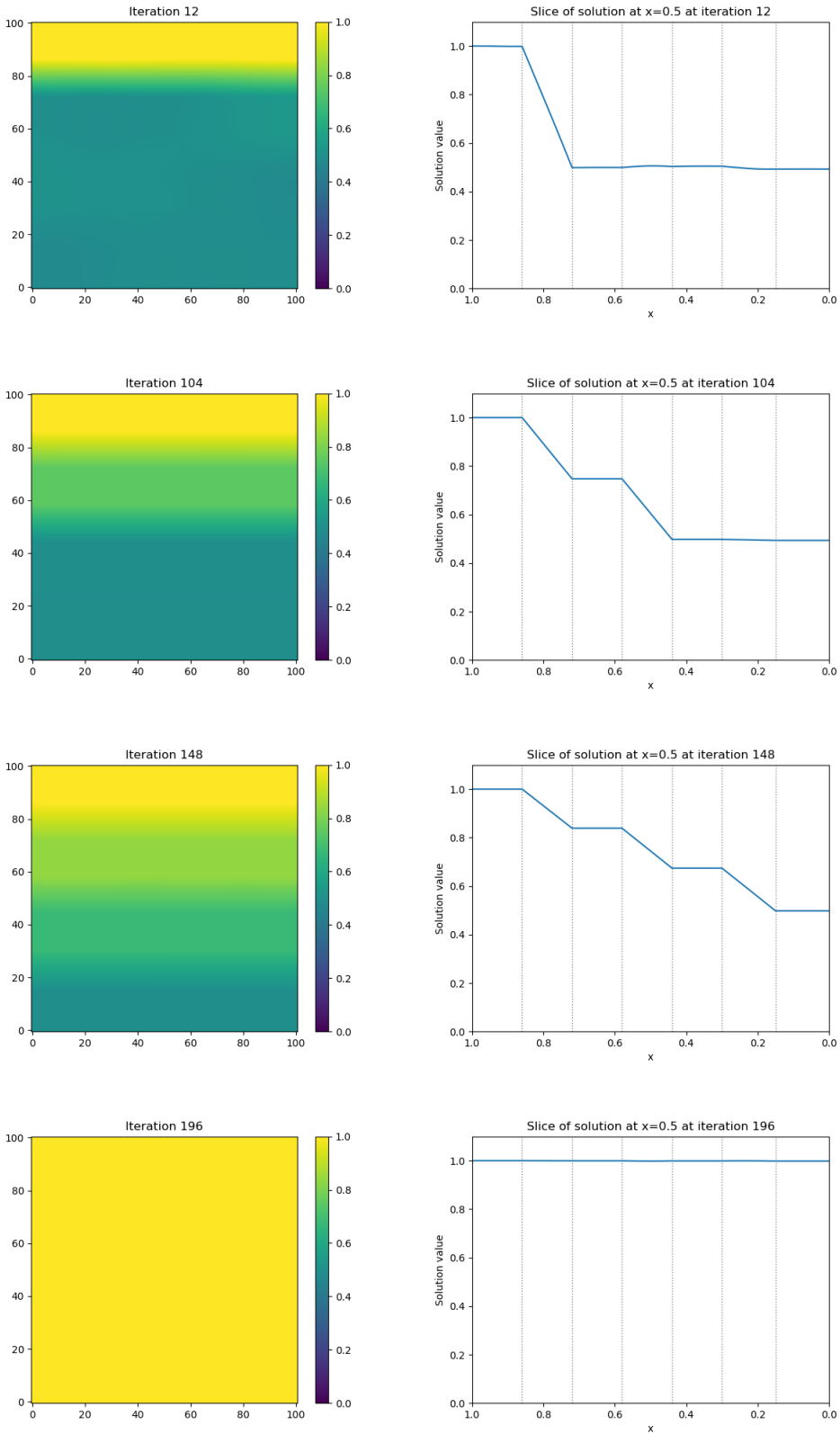
5.2.1 Convergence analysis

To better understand the convergence of these methods, a closer look at the convergence is required. In Figure 17, the solution on the entire domain and in a slice at $x = 0.5$ are shown at select iterations. After 12 iterations, the solution is constant in the top sandstone layer, linear in the shale layer below, and roughly constant in the rest of the domain. In the next 80 iterations, the solution barely changes, as at iteration 93 the solution is still roughly the same, while the residual still decreases between iterations. The difference in solution between iteration 12 and iteration 93 can be seen in Figure 16. At iteration 77, the local minimum, the relative residual reaches a value of order 10^{-9}, which is nearly at the relative tolerance.



(a) Solution in the entire domain (b) Slice of the solution at $x = 0.5$

Figure 16: Evolution of the solution using the ICCG(0) method, between iteration 12 and 93



(a) Solution in the entire domain (b) Slice of the solution at $x = 0.5$

Figure 17: Evolution of the solution using the ICCG(0) method, at select iterations

After the 93rd iteration, the solution changes more quickly over the next 11 iterations, reaching a new solution at iteration 104, which, again, stays the same until iteration 141, after which it changes again. This whole process repeats itself 3 times, until finally the solution converges to the correct solution. The solution reaches some kind of plateau when the information reaches a shale layer, and that the information is slow to travel through the shale layers.

In [2], Vuik, Segal and Meijerink also make note of this behaviour for a similar problem with 7 layers. They perform an analysis of the eigenvalues of the system. One would expect there to be at least as many small eigenvalues of roughly order 10^{-7} (the value of μ_{shale} as there are internal shale nodes, as the entries in the corresponding rows of matrix A are of order 10^{-7} relative to the rest of the matrix entries. However, in their analysis they find 3 extra small eigenvalues. Additionaly, they show that for a problem with n layers and sufficiently small values of ε, the number of eigenvalues of the diagonally scaled matrix $D^{-\frac{1}{2}}AD^{\frac{1}{2}}$ only has n eigenvalues of $\mathcal{O}(\varepsilon)$. These remaining small eigenvalues are associated with the jump in coefficients between the layers. More precisely, the number of small eigenvalues is equal to the number of sandstone (high permeability) layers between shale (low permeability) layers [2]. An accurate solution can only be computed when all these small eigenvalues are 'discovered' by the solver. This discovery of a small eigenvalue is analogous with a bump in the residual plot.

In Figure 18, the residual plot for a problem with 11 layers is shown. This plot has 5 bumps, as is expected from the previous analysis, since the problem has 5 high permeability layers in between low permeability layers.

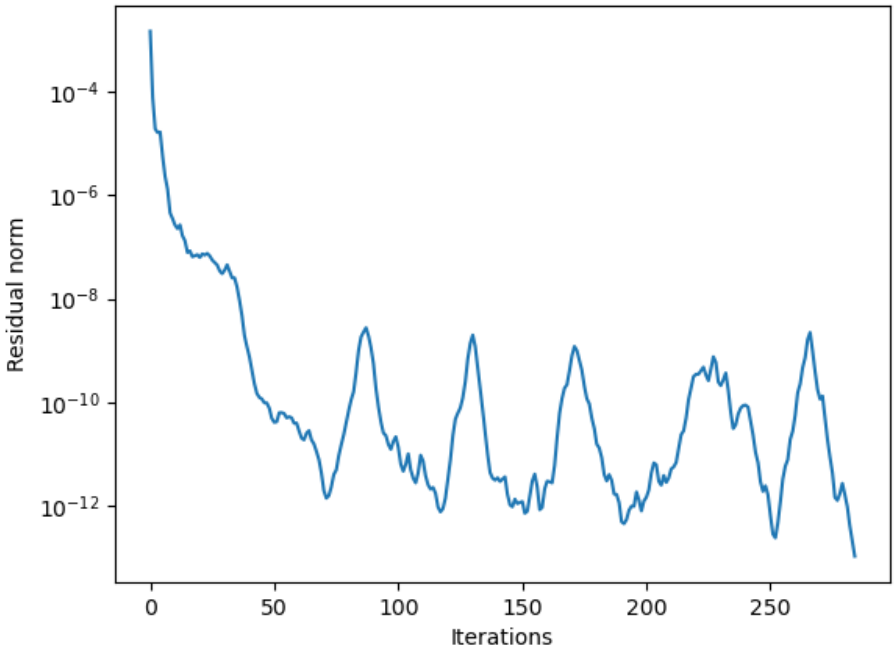


Figure 18: Residual plot for the ICCG(0) method for a problem with 11 layers

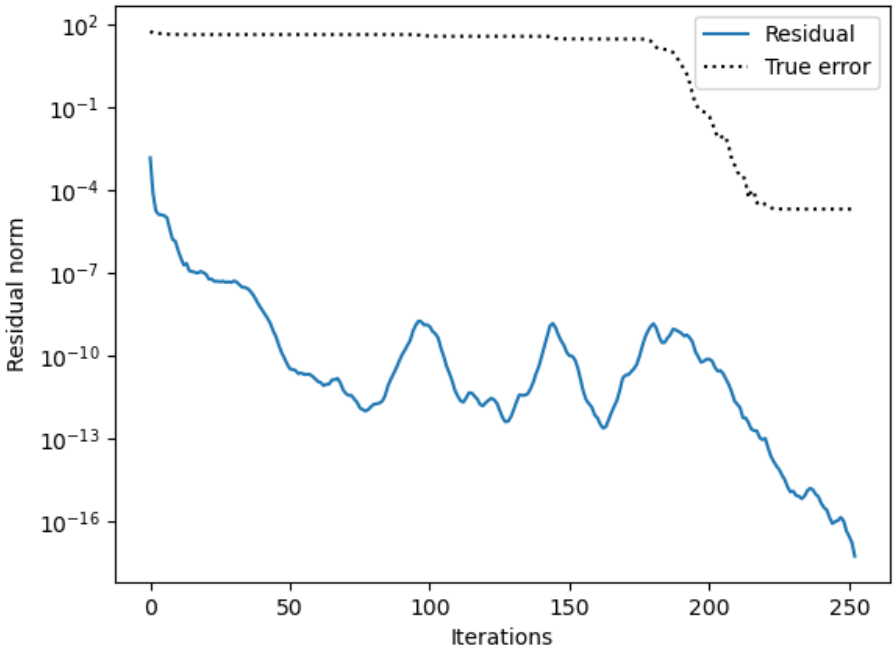


Figure 19: Residual and error norm plots for the ICCG(0) method with relative tolerance 10^{-14}, 7 layers

Another important observation is that the residual is not a reliable stopping criterion for the numerical solver. As noted before, a relative tolerance of 10^{-9} would not be enough to find the right solution for this problem, as the solver would terminate before the first small eigenvalue is discovered. In Figure 19 the error norm is shown alongside the residual norm. The error norm hardly decreases until all small eigenvalues are discovered. The error norm only starts to decrease once the residual plot has reached the peak of its third bump.

Additionally, there is a limit to the improvement of the true error. This effect is illustrated in Figure 19, where the problem is solved with the relative tolerance set to 10^{-14}. The true error improves by order 10^{-7}, and the accuracy does not improve any further. This is also noted by Vuik, Segal and Meijerink [2], with a remark that the true error does not improve by more than $c\kappa_2(A)u||\mathbf{x}-\mathbf{x}_0||_2$, where c is a small constant and $u \approx 10^{-16}$ is the (relative) machine precision. To the authors knowledge, there is no proof that shows such a bound, but experimental observations support the remark.

5.3 Deflation

The deflation method is explained in Section 3.3. This method is very effective at removing the effect from a small amount of problematic eigenvalues. As seen in the convergence analysis in the previous section, classic preconditioning methods such as Jacobi and IC(0) preconditioning do not remove all small eigenvalues from the system. For the model problem consisting of 7 layers with 3 shale layers, three small eigenvalues remain in the system after preconditioning. Therefore, the deflation method is very attractive for removing these few small eigenvalues.

Ideally, the columns of restriction matrix Z would be the eigenvectors corresponding to the small eigenvalues. However, computing these eigenvectors is very costly and not viable. There are also large computational costs associated with approximating the eigenvectors. However, as the small eigenvalues are associated with the jump in coefficients between domains, it is also possible to use subdomain deflation to construct the restriction matrix Z. This method is also described in Section 3.3. An attractive property of this method is that it does not require computing or approximating any eigenvectors. In fact, constructing the restriction matrix for subdomain deflation is almost trivial.

5.3.1 Choice of subdomains

An important part of subdomain deflation is the definition of the subdomains. A logical choice for the subdomains are the layers. However, due to the finite element discretisation, there is a row of discretisation points at the interface that is part of both adjacent layers. Because each element contains its own material type, there must by definition be points on the interface between two layers of a different material type.

In section 3.3, the restriction matrix is defined with each point belonging exclusively to one subdomain. Each column of the restriction matrix corresponds to a subdomain. Every value in the deflation matrix is 1 if the point corresponding to its row is part of the subdomain corresponding to its column, and 0 otherwise. However, it is also possible to let a point belong to two subdomains. Recall that the index sets corresponding to the subdomains Ω_j are defined as $\mathcal{I}_j = \{i \in \mathcal{I} | \mathbf{u}_i \in \Omega_j\}$, where $\mathcal{I}$ is the index set of all points in the entire domain Ω. The the restriction matrix is defined as:

$$z_{i,j} = \begin{cases} 1, & i \in \mathcal{I}_j, \\ 0, & i \notin \mathcal{I}_j. \end{cases}$$

Now, let Γ_j denote the interface of subdomain Ω_j. Then the index set corresponding to the subdomain interface is $\mathcal{I}_{\Gamma_j} = \{i \in \mathcal{I} | \mathbf{u}_i \in \Omega_j\}$. The restriction matrix can be defined as follows:

$$z_{i,j} = \begin{cases} 1, & i \in \mathcal{I}_j \setminus \mathcal{I}_{\Gamma_j}, \\ \delta_j & i \in \mathcal{I}_{\Gamma_j}, \\ 0, & i \notin \mathcal{I}_j, \end{cases}$$

where δ_j is some value. It is common, but not necessary, to use $\delta_j = 0.5$ for all j.

To find the correct way to handle the interface points, four different options for handling the interface points are considered. Let $\mu_j \in \{\mu_{sandstone}, \mu_{shale}\}$ denote the permeability value of the j-th layer for $j = 1, 2, \ldots, 7$.

1. $\delta_j = 1$ if $\mu_j = \mu_{sandstone}$, $\delta_j = 0$ otherwise
2. $\delta_j = 1$ if $\mu_j = \mu_{shale}$, $\delta_j = 0$ otherwise

3. $\delta_j = 0.5$ for all j

4. $\delta_j = \mu_j/(\mu_{sandstone} + \mu_{shale})$ for all j

Method 1 corresponds with assigning the interface points to the adjacent sandstone layer. Method 2 corresponds with assigning the interface points to the adjacent shale layer. Method 3 simply sets all interface points to $0,5$ in both layers, while method 4 scales the influence of the interface points in each layer according to the value of the permeability in each layer.

The 4 methods of dealing with the interface points are tested using the ICCG(0) method combined with deflation. The convergence rate for each of these choices is shown in Table 3. Note that method 1 and 4 converge the fastest with exactly the same number of iterations. Consider a point on the interface of two layers. The coefficients in the row of the matrix corresponding to this point will be of roughly the same order as the coefficients of the layer with the largest coefficient. In this case that is the adjacent sandstone layer. Therefore, it makes sense that the interface points should belong to the subdomain corresponding with the sandstone layer, as all coefficients are roughly the same order of magnitude. The reason that method 4 gives similar convergence is easy to see: as $\mu_{shale} << \mu_{sandstone}$, influence of the interface points in the subdomains corresponding with shale layers is negligible. As both methods have the same effectiveness, method 1 is preferred, as it makes construction of the restriction matrix rather simple.

However, for a general domain it might be more useful to use method 4. In model problem 2, the contrast in coefficients between two layers is always the same, and the contrast is large. In a practical application a range of coefficients can be present in a domain. In that case, the influence of the smaller coefficient will be larger in method 4. This method provides more of a 'black box' than method 1.

Interface method	#iterations
1	79
2	126
3	163
4	79

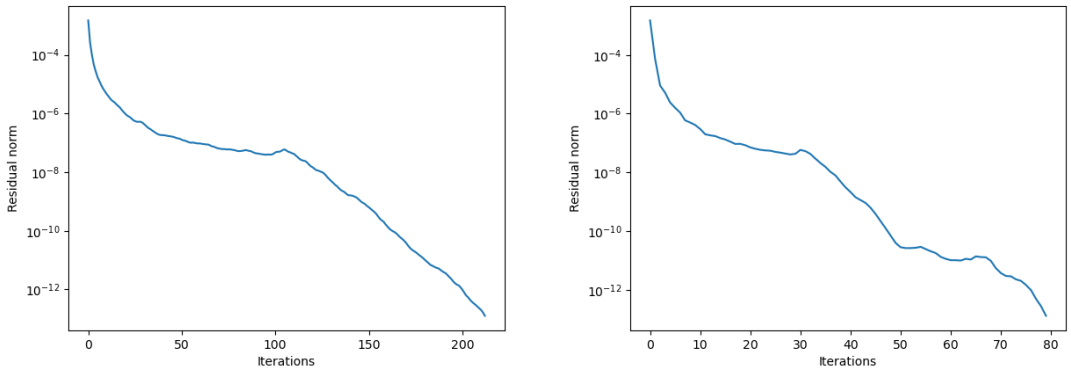
Table 3: Number of iterations for the ICCG(0) method with deflation, different definitions of the subdomains

5.3.2 Conjugate gradient and preconditioning

The deflation method on its own is not enough to improve the CG method, as it is only able to deal with the small eigenvalues in the system associated with the jump in variables. In fact, solving the problem with the CG method combined with deflation takes 2728 iterations. Therefore, the deflation method must be combined with some kind of preconditioning to improve the rate of convergence.

Using the method chosen in the previous section for defining the subdomains, deflation is used in combination with both the JCG and the ICCG(0) method. These methods will be denoted as DJGC and DICCG(0) respectively. The residual plots for both preconditioners can be seen in Figure 20. In these figures it is clearly visible that deflation is able to effectively remove the effect of the small eigenvalues corresponding to the jumps in coefficients. The residual plot follows roughly the same curve as the JCG and ICCG(0) methods without deflation at the start. However, the three bumps are removed from the residual plot. Checking the solution of these methods verifies

that the correct solution is found. For the DICCG(0) method, the number of iterations is 82, for the JCG method the number of iterations is 220. Both are significant improvements when compared to their convergence rate without deflation.



(a) DJCG (b) DICCG(0)

Figure 20: Residual norm plot for the DJCG and DICCG(0) methods

5.3.3 Convergence analysis

In Figure 21 the residual and error norm are plotted for the DICCG(0) method. In Section 5.2.1, the same plot is shown for the ICCG(0) method (i.e. the same method without the use of deflation) in Figure 19. Recall that the true error does not improve for the ICCG(0) method until after all small eigenvalues had been discovered by the solver. For the DICCG(0) method, the error starts improving immediately, and follows roughly the same curve as the residual norm, albeit with a different order of magnitude. This means that the relative residual is a reliable stopping criterion for the DICCG(0) method, as an improvement of the residual norm coincides with an improvement in the error norm. Note that again, the error norm does not improve by more than an order of 10^{-7}.

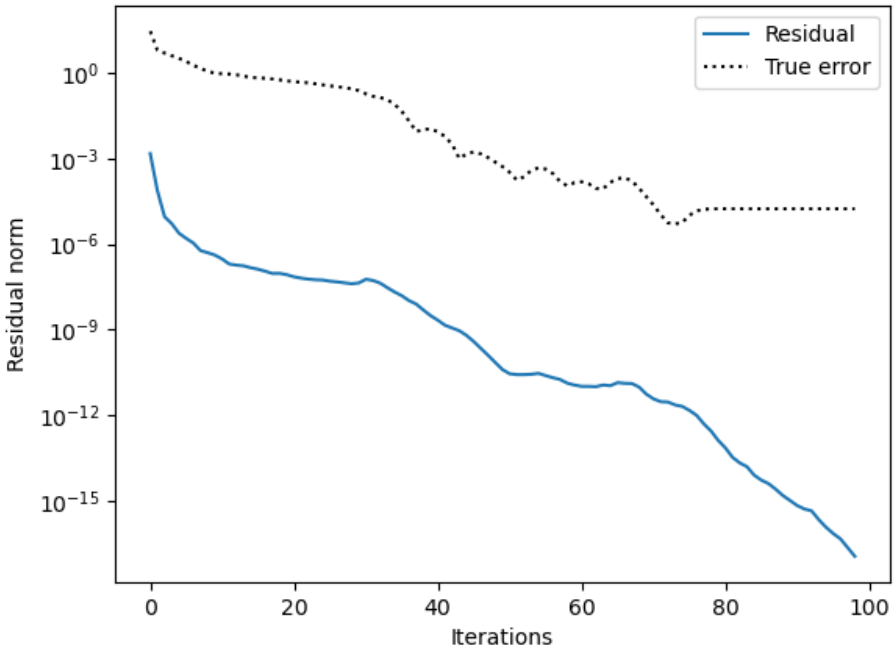


Figure 21: Residual and error norm plots for the DICCG(0) method, with relative tolerance 10^{-14}

5.4 Restricted additive Schwarz

The restricted additive Schwarz (RAS) preconditioner is described in Section 3.4. It is a domain decomposition method. The domain Ω is divided into a number of overlapping subdomains Ω_j and corresponding non-overlapping subdomains $\tilde{\Omega}_j$. Application of the preconditioner consists of computing a correction on each subdomain Ω_j, restricting this correction to $\tilde{\Omega}_j$, and adding the restricted correction to the iterative solution.

Recall that the RAS preconditioner is defined as $B_{RAS} = \sum_i \tilde{R}_i^T A_{\Omega_i}^{-1} R_i$. Computing the inverse A_{Ω_i} directly can still be expensive if the matrix is large. Therefore, the correction is computed using an iterative method such as CG. Here it becomes clear why this method is considered for this problem. The CG method is not very effective for solving the problem without preconditioning and deflation methods. The large contrast in coefficients is the main cause of the poor conditioning of the linear system. Additionally, information is slow to travel through layers with a small coefficient. The subdomains for the RAS method can be chosen to coincide with the layers. Within one layer, the coefficients are constant, therefore the subdomain matrix A_{Ω_i} should be conditioned much better than the system matrix A, as it does not contain any big jumps in the coefficients. Thus, computing a correction on a subdomain with CG should be efficient. This is the main motivation for using the RAS method.

5.4.1 Choice of subdomains

As stated previously, the main idea is to have the subdomains coincide with the layers. As there are 7 layers in the model problem, 7 subdomains are defined. The RAS method requires the definition of an overlapping set of subdomains Ω_i and a non-overlapping set of subdomains $\tilde{\Omega}_i$ for $i = 1, 2, \ldots, 7$. The most common way to define these subdomains is to start with a non-overlapping set of subdomains. The overlapping subdomains are then defined by extending the non-overlapping by a number of points in any direction. It is common to use an overlap of 1, but it is possible to make the overlap larger, although a larger overlap requires more computational power, and is undesirable in a parallel environment, as it increases communication costs.

However, for the model problem the layers themselves define a set of overlapping subdomains. Recall that due to the finite element discretisation there is a row of discretisation points on every interface between two layers. These discretisation points do not belong to either layer, but are corner points of elements belonging to either layer. Therefore, the layers define the overlapping set of subdomains. Note that this is a set of subdomains with minimal overlap. Any two subdomains only share at most one row of discretisation points. An example of the definition of the subdomains with overlap in three layers is shown in Figure 22.

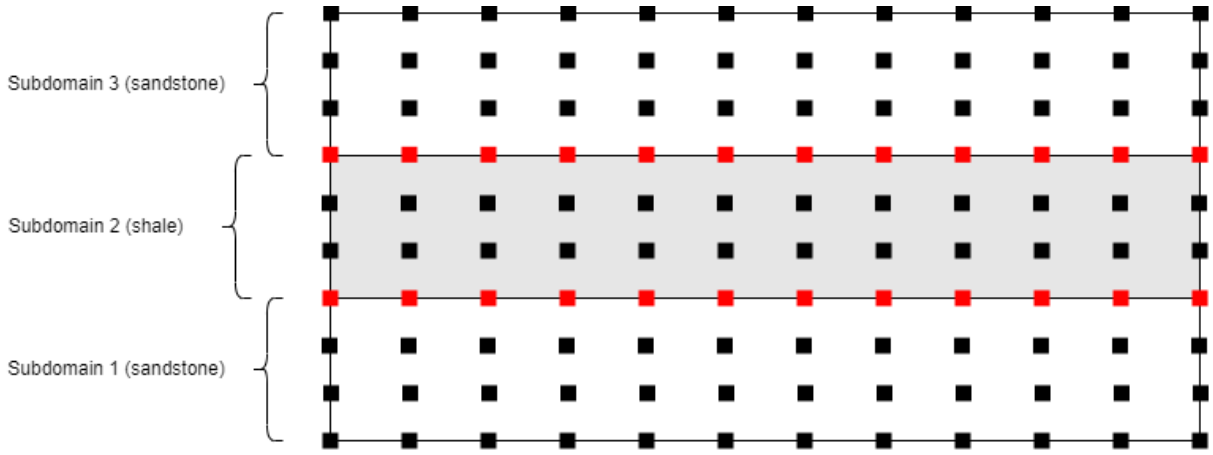


Figure 22: Example definition of the overlapping set of subdomains for the RAS method

The definition of the subdomains without overlap is less obvious at first glance. The discretisation points on the layer interfaces do not belong to either layer. However, in the definition of the non-overlapping subdomains they must be assigned to one of the two layers. Consider a discretisation point on the interface between two layers. Due to the problem definition, one of these layers is a shale layer and the other is a sandstone layer. The discretisation point is the corner point of at least one shale element and one sandstone element. Therefore, the values in the corresponding row of matrix A will be influenced by both element matrices. The values in the sandstone element(s) and shale element(s) will be of order 1 and order 10^{-7} respectively (relatively to each other). Therefore, the largest values in that row of the matrix will be of order 1.

The matrix rows corresponding to internal shale discretisation points will have values of order 10^{-7}, while the matrix rows corresponding to internal sandstone discretisation points will have values of order 1. As it is desired to create subdomains without a large change in coefficients, it makes the most sense to assign the interface nodes to the adjacent sandstone layer. Therefore, the subdomains without overlap are defined as follows:

$$\tilde{\Omega}_i = \begin{cases} \{\mathbf{u}_j | \mathbf{u}_j \in \Omega_j \setminus \Gamma_j\} & \text{if } \Omega_i \text{ is a shale layer,} \\ \Omega_i & \text{if } \Omega_i \text{ is a sandstone layer.} \end{cases}$$

An example of the definition of the subdomains without overlap in three layers is shown in Figure 23 Numerical experiments confirm that this is the correct way to define the non-overlapping subdomains, as other definitions do often not lead to convergence.

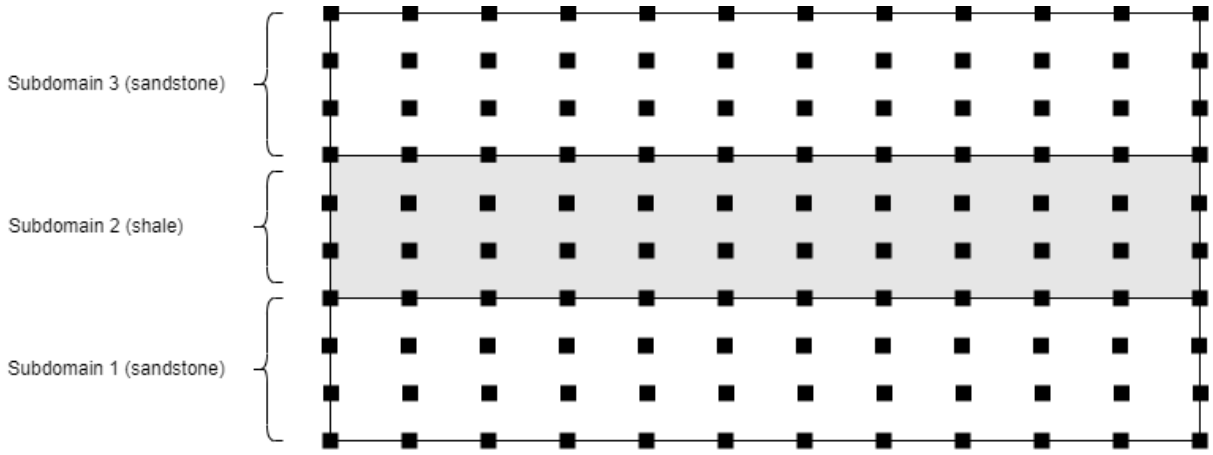


Figure 23: Example definition of the non-overlapping set of subdomains for the RAS method

5.4.2 Computing the subdomain corrections

In the RAS method, in every iteration a correction is computed on every subdomain. As mentioned before, this could be done with a direct method, as the linear systems corresponding to the subdomains are much smaller than the linear system for the whole domain. However, for large subdomains a direct method would still be inefficient. Additionally, it is not required for the solution of the subdomain solve to be very precise. In the model problem all information comes from the top boundary, and needs to be passed through the entire domain before an accurate solution can be found. The RAS preconditioner adds a correction to each subdomain at every iteration. This correction helps to propagate the information from the top boundary through the domain much more efficiently. The correction does not need to be incredibly accurate to achieve this. Therefore the choice is made to use iterative methods for the subdomains as well. As the matrix A is SPD, the subdomain matrices A_{Ω_i}, $i = 1, 2, \ldots, 7$ are SPD as well. Therefore a preconditioned CG method can be used to compute the corrections on the subdomains.

For the preconditioner we consider the two methods used before in this thesis: the JCG and the ICCG(0) methods. Previously, the ICCG(0) method has consistently outperformed the JCG method both in terms of number of iterations and CPU time. Since each subdomain consists of a layer with constant coefficients, the expectation is that the ICCG(0) method will perform better on the subdomains. Numerical experiments confirm that this is true. Note that the number of iterations required to find a solution on the entire domain is the same regardless of the method used on the subdomains. Both the JCG and the ICCG(0) method compute a correction of the same accuracy. However, the JCG method requires more iterations and CPU time to compute the correction on each subdomain.

The termination criteria for the subdomain solvers can be relaxed when compared to the solver for the entire domain. Numerical experiments show that a relative tolerance of 10^{-5} and absolute tolerance of 10^{-18} do not slow down the rate of convergence, but provide a significant decrease in CPU time needed to reach convergence when compared to smaller tolerances. If these tolerances are made larger, the convergence rate becomes worse.

5.4.3 Conjugate gradient

The problem is solved using the CG method with RAS preconditioning, this will be referred to as the RASCG method. Recall that the RAS preconditioner is asymmetric. The preconditioned

conjugate gradient method is defined for an SPD matrix A with an SPD preconditioner matrix M. Therefore, one might expect that the CG method with RAS preconditioning will not converge. However, the amount of asymmetry introduced by the preconditioner is limited, since there are only a small number of subdomains. The effectiveness of the RASCG could reduce significantly for a larger number of subdomains, as a larger amount of asymmetry would be present in the system And, as will be shown in this section, the CG method with RAS preconditioning does actually converge for this problem. It has been shown by Bouwmeester, Dougherty and Knyazev that the PCG method can converge with a non-symmetric geometric multigrid preconditioner [16].

Convergence is reached in 10 iterations. This is significantly faster than the other methods discussed. However, it is important to keep in mind that at every iteration an iterative solver is used to compute the correction on each subdomain. Therefore, while convergence is much faster, this method does not necessarily perform better in terms of CPU time, as it is possible that a lot of work is required for the subdomain solvers.

In Figure 27, the residual and error norm plots are shown. Note that just like the ICCG(0) method, the error norm does not decrease until the final few iterations, even though the residual norm decreases very quickly. Afterwards, the residual norm stays roughly the same for a few iterations, before improving in the final iterations. Similarly, the relative residual norm is not a very reliable stopping criterion, as it only assures an accurate solution if the tolerance is set small enough.

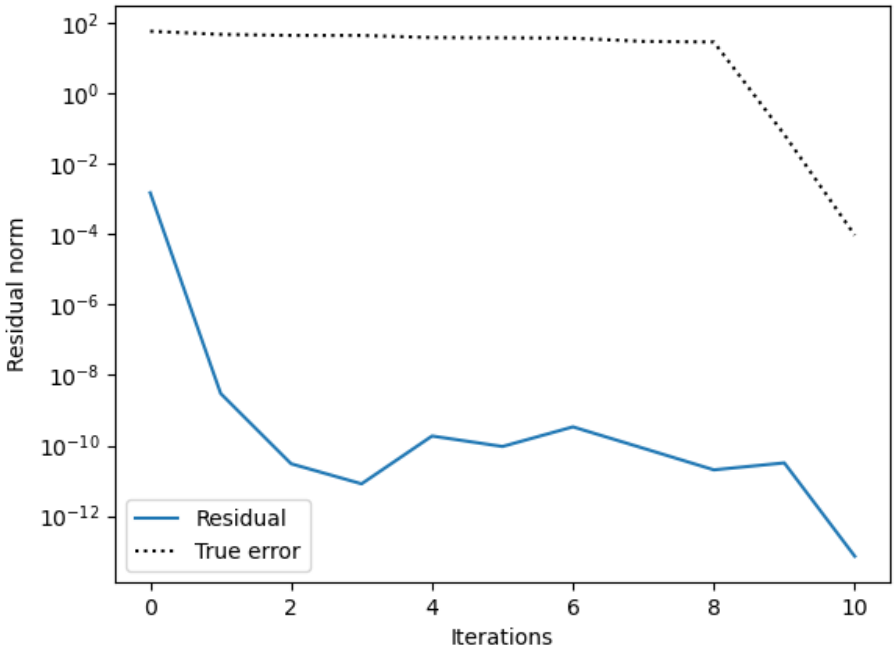


Figure 24: Residual and error norm plots for the CG method with RAS preconditioning

The solution at every iteration can be seen in Figure 25. At iteration 0 the starting guess is visible. After that it is clear that this method is able to propagate the solution through each layer in one or two iterations, regardless of the material type of that layer.

In Figure 26, a slice of the solution at $x = 0.5$ is shown after every iteration. This gives a

clear image of how the solution changes through the iterations. Interestingly, the solution follows roughly the same pattern as the solution of the ICCG(0) method. In particular, the solutions found at iteration 3, 5 and 7 are very similar to the solutions found by the ICCG(0) method (see Figure 17). The number of iterations required to find these solutions is reduced greatly. Due to the corrections on the subdomains, the information from the top boundary is able to travel through the a layer in 1 or 2 iterations.

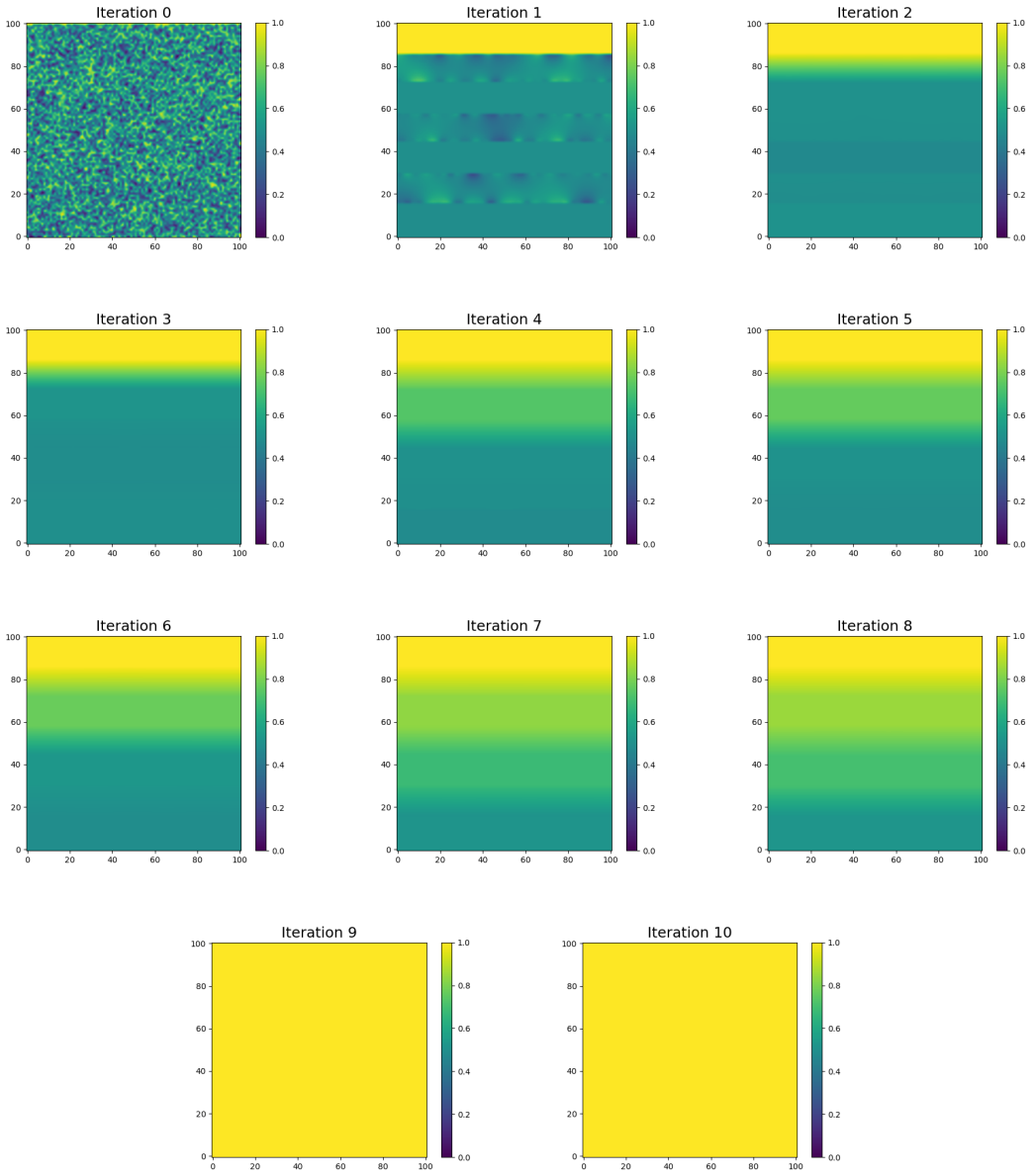


Figure 25: Solution at every iteration using the CG method with RAS preconditioning

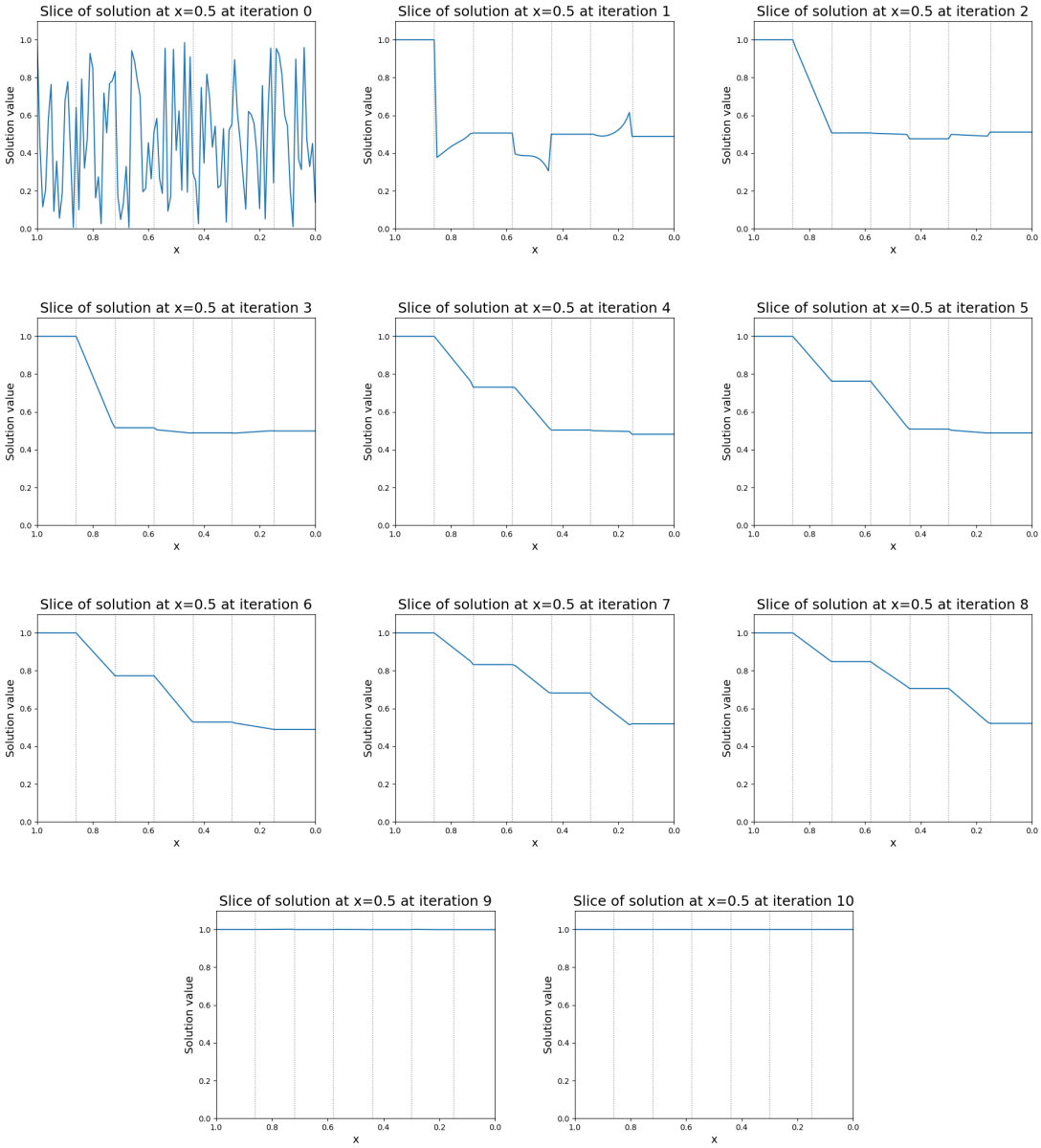


Figure 26: Slice of the solution at every iteration at $x = 0.5$, using the CG method with RAS preconditioning

5.5 Combining deflation and RAS

In the previous two section the deflation and RAS methods are discussed. As deflation and a preconditioner can be combined, it is possible to use both methods at the same time. This method will be referred to as the DRASCG method. This method converges even more quickly than the CG method with RAS preconditioning. With a relative tolerance of 10^{-14} the method converges in 7 iterations. Note that if the tolerance is larger, for example 10^{-10}, the method will stop after only 4 iterations, since the residual is already small. However, the true error has only reduced by a factor of 10^{-4} at this point. This is likely due the extremely low number of iterations that have been performed. The smaller value for the relative tolerance is required to perform more iterations, as PETSc does not have an option to set a minimum number of iterations. With the extra iterations the error is able to be improved by an order of 10^{-7}.

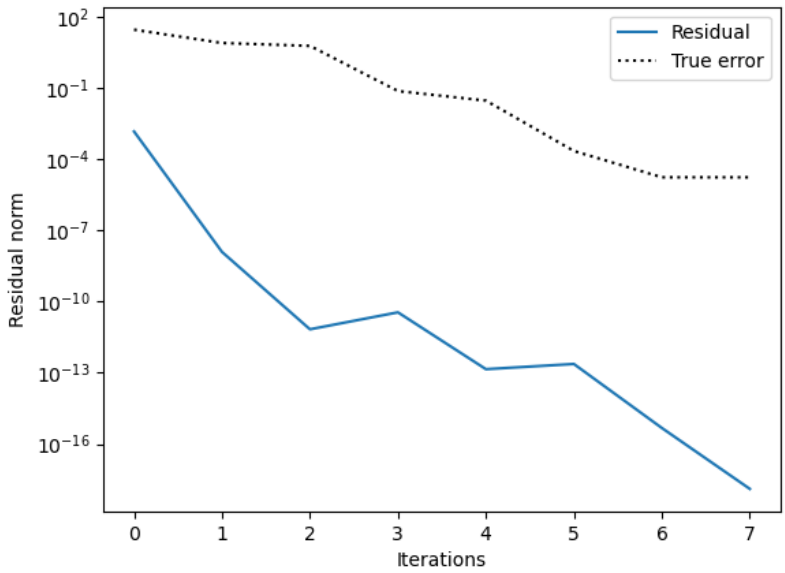


Figure 27: Residual and error norm plots for the CG method with RAS preconditioning and deflation, with relative tolerance 10^{-14}

5.6 Performance

In this section, the performance of the different methods discussed will be compared, both in terms of iterations and in term of CPU times. These tests are all performed sequentially. Note that these tests were performed on a personal device. Effort has been made to create an isolated environment. However, it can not be ruled out that these timing results have been influenced by variance in the performance of the device.

For all methods, the same stopping criterion has been used: the a relative tolerance of 10^{-10}. The only exception is the DRASCG method. For this method, the relative tolerance is set at 10^{-14} to ensure that the same accuracy of solution is reached. The results are summarised in Table 4.

Note that

Method	#iterations	CPU time (s)
CG	9163	4.754
JCG	726	0.457
ICCG(0)	218	0.252
DJCG(0)	220	0.204
DICCG(0)	86	0.113
RASCG	10	0.424
DRASCG*	7	0.259

Table 4: Number of iterations and CPU times for different methods. *For the DRASCG method the relative tolerance is set at 10^{-14}.

For this problem, the RASCG and the DRASCG methods have the smallest number of iterations. It should be noted that these methods do required solving several smaller linear systems at every iterations. Another method requiring relatively few iterations is the DICCG(0) method.

In terms of CPU times, the DICCG(0) method clearly outperforms all other methods. The extra work required for the RASCG method and DRASCG method results in slower CPU times.

5.7 Influence of the contrast in coefficients

Model problem 2 has a large contrast in coefficient as its key feature. This contrast has large impact on the convergence rate of the different solvers used, and also affects the amount by which the true error can be improved. In this section, the effect of the contrast is investigated by comparing the previously discussed methods when the contrast in the coefficients is different.

To perform these tests, model problem 2 is considered with a variable value for μ_{shale}, while the value of $\mu_{sandstone}$ remains fixed at 1. Note that it does not matter which value is varied, the main thing that matters is the contrast between the coefficients μ_{shale} and $\mu_{sandstone}$. Since the value of $\mu_{sandstone}$ is equal to 1, the value of μ_{shale} also represents the order of the contrast between the coefficients.

The large contrast in model problem 2 leads to a large number of iterations for the CG method. To better understand this effect, the convergence rate of the CG method is compared to the ICCG(0), DICCG(0), RASCG and DRASCG methods for different values of μ_{shale}. The JCG and DJCG methods are omitted here, since the behaviour is similar to the ICCG(0) and DICCG(0) methods respectively. Note that the case where $\mu_{shale} = 1$ is equal to the problem on a uniform domain. The results are shown in Table 5.

μ_{shale}	CG	ICCG(0)	DICCG(0)	RASCG	DRASCG
1	380	107	81	118	142
10^{-1}	657	129	94	1025	178
10^{-2}	1376	145	94	86	42
10^{-3}	2931	155	79	33	22
10^{-4}	5114	164	79	19	15
10^{-5}	6854	176	79	14	11
10^{-6}	8332	201	79	12	8
10^{-7}	9163	218	79	10	7

Table 5: Number of iterations for several methods for different values of μ_{shale}

The CG method without preconditioning shows the expected behaviour. The number of iterations increases greatly when the contrast between the coefficients grows larger. This confirms that the CG method has limited usability without the use of preconditioning. The ICCG(0) also shows an increase in the number of iterations, although it is much less pronounced.

The DICCG(0) shows the most consistent behaviour of all considered methods. For values of μ_{shale} ranging between 10^{-3} and 10^{-7} the rate of convergence is exactly the same. For larger values of μ_{shale} the number of iterations is larger, but only by a small margin. The method is robust regardless of the contrast between the coefficients. It should be noted that the discretisation points on the layer interfaces are assigned to a subdomain based on a problem with a large contrast in coefficients. For a smaller contrast, the convergence rate might be improved by using interpolation to handle the interface points. When the contrast is small, the improvement in convergence rate of the DICCG(0) compared to the ICCG(0) rate is small, and the extra cost associated with the subdomain deflation method is most likely not worth the effort. In [17], it is noted that this type of deflation has potential for commercial applications for a contrast of at least order 10^{-4}.

The RASCG method shows a better convergence rate when the contrast is larger. While the convergence DICCG(0) method remains stable for all values of μ_{shale} smaller than 10^{-3}, the conver-

gence rate of the RASCG method keeps improving. Upon closer inspection, it becomes clear that the method is less effective at propagating the information through the domain when the contrast is smaller. Specifically, the solution propagates slower through the subdomains corresponding to the sandstone layers. For $\mu_{shale} = 10^{-7}$, the solution is almost exact in the top sandstone layer after 1 iteration. For $\mu_{shale} = 10^{-1}$, this takes several iterations. A slice of the solution after 1 iteration can be seen in Figure 28. The method is much less effective at finding the correct solution in the top layer than for a large contrast in coefficients (Figure 26). Since the relevant information is coming from the top boundary, the values in the rest of the domain are 'incorrect' compared to the actual solution. When the contrast is small, this incorrect information is propagated back towards the top of the domain more quickly, especially in the CG iteration after application of the preconditioner, thus slowing down the rate of convergence of the RASCG method.

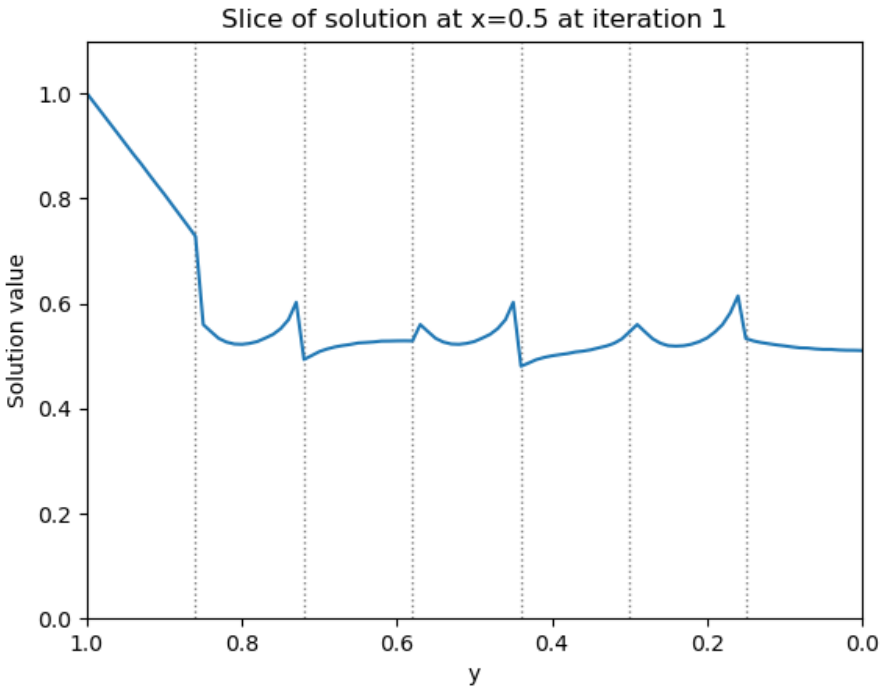


Figure 28: Residual norm plots for the RASCG method with $\mu_{shale} = 10^{-1}$

In a commercial application, it is likely that there is a good starting guess available, therefore this effect should be reduced. However, it is still clear that the RASCG method performs less well for a smaller contrast, especially considering the extra work required for the application of the preconditioner. The extremely large amount of iterations required for the case that $\mu_{shale} = 10^{-1}$ is likely a result of 'overshooting' of the subdomain corrections, as close inspection of the iterative solutions shows that the information is propagated through the entire domain after roughly 30 iterations. The remaining iterations are used to reach the desired accuracy.

The DRASCG method shows similar behaviour to the RASCG method, working better for a larger contrast in the coefficients. Both methods are most suited for problems with a large contrast, otherwise the work required for the RAS preconditioner outweigh the benefit of faster convergence.

Another effect of the contrast in coefficients is the achievable accuracy of the solution. It has

been noted in the previous sections that the improvement of the error is limited to an order of 10^{-7}. This is a direct result of the contrast in coefficients, as can be seen in Table 6, where the limit of the relative error norm is shown for different values of μ_{shale} for the DICCG(0) method. Note that these values are of the same order for the different methods. It is clear that the achievable accuracy of the solution is significantly worse for a larger contrast in the coefficients.

μ_{shale}	Limit on error improvement
1	$5.22 \cdot 10^{-14}$
10^{-1}	$9.84 \cdot 10^{-14}$
10^{-2}	$4.66 \cdot 10^{-13}$
10^{-3}	$6.95 \cdot 10^{-12}$
10^{-4}	$1.60 \cdot 10^{-10}$
10^{-5}	$1.85 \cdot 10^{-10}$
10^{-6}	$2.05 \cdot 10^{-8}$
10^{-7}	$1.73 \cdot 10^{-7}$

Table 6: Limit on the error improvement for the DICCG(0) method for different values of μ_{shale}

Along with this limit on the relative error norm, the relative residual is observed to be an unreliable stopping criterion for several of the discussed methods. Only the methods that use the deflation method can use the relative residual as a reliable stopping criterion. For the other methods, the correct solution is not found if the relative tolerance is not tight enough.

Consider the ICCG(0) method with a relative tolerance of 10^{-9}. For model problem 2, the method converges after 75 iterations, to the incorrect solution. A slice of the solution at $x - 0.5$ is shown in Figure 29. This type of solution is typical for all methods that do not use deflation when the relative tolerance is not tight enough. The solution is constant in the top layer, linear in the second layer from the top, and constant again through the rest of the domain. The residual norm does not recognise that the solution is incorrect, even though the error is still large. Recall that the residual $\mathbf{r}^k$ at iteration k is equal to A multiplied with the error $\mathbf{e}^k = \mathbf{u}^k - \mathbf{u}$, where $\mathbf{u}$ is the exact solution.

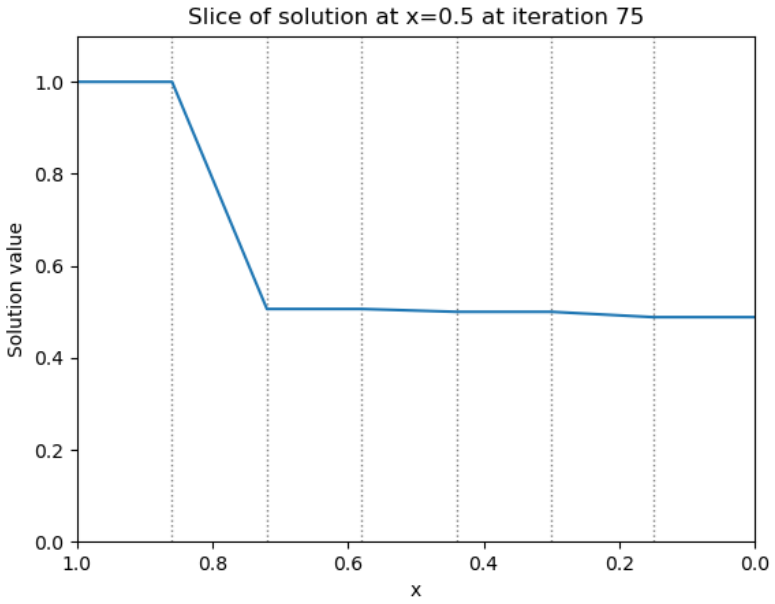


Figure 29: Residual norm plots for the ICCG(0) method with relative tolerance 10^{-9}

The error in the second layer (and all shale layers) is reduced by a factor 10^{-7}, as the values in the corresponding rows of A are of that order compared to the other rows. The error in the lowest 5 layers is not detected by the residual, as it is constant. Therefore the residual only recognises the error in the second layer, and this error is reduced greatly due to the contrast in coefficients. If the tolerance is not tight enough, this solution is seen as satisfactory and the iterative process is terminated, even though the solution is not accurate in the slightest.

The deflation method does not suffer from this problem, as the CG method is used to solve the deflated system $PA\tilde{\mathbf{u}} = P\mathbf{f}$ instead of the regular linear system.

6 Parallelisation

In the previous sections all computations were performed sequentially. In practical applications, the number of nodes is often large, and it is not feasible to solve a given problem on a single processor. Therefore, the methods discussed in the previous sections are tested in a parallel environment. In particular, the deflation method and the RAS method are compared. The goal is to find out how well they perform in a parallel environment, and what kind of speedup can be achieved.

For all parallel timing tests, the Snellius supercomputer is used. For parallel timing tests, the domain is divided into 1000×1000 square elements, instead of the discretisation into 100×100 elements in the definition of model problem 2. For the parallel results to be meaningful, the number of discretisation points needs to sufficiently large. For the 100×100 discretisation, the number of discretisation points is 10100 (there are 101 discretisation points in each direction, but the row on the Dirichlet boundary is eliminated).

For such a small problem size the computation times are small, especially on the Snellius supercomputer, as the processors are more powerful than those in a personal device. In parallel, the processors need to exchange data along the interface of their ownership range after every iteration. This communication causes a certain amount of overhead, as it introduces some latency when data is transferred between processors. If the time spent performing computations is small, the communication overhead is relatively large, and therefore the amount of speedup that can be obtained is limited. For the 1000×1000 discretisation, the number of discretisation points is 1001000. Due to the larger number of discretisation points, the amount of time required to perform computations becomes relatively larger, since the number of discretisation points is scaled by a factor of 100, while the number of data points that are communicated along a boundary is only scaled by a factor of 10.

In Section 6.1, the parallel performance will be analysed for up to 7 processors, using a physics based data distribution. Then, in Section 6.2, the comparison is made between a data based distribution of data points and a physics based distribution for a larger number of processors.

6.1 Parallel performance for a small number of processors

In this report, a physics based distribution has been used for defining the subdomains, both for the deflation method and the RAS method. That is, the subdomains are chosen to coincide with regions in the domain with the same coefficients. In this section, parallel performance tested for a small number of processors, such that the number of processors does not exceed the number of layers. As a result, each subdomain can fall within the ownership range of a single processor. The data is distributed in such a way that each subdomain corresponding to a layer falls within the ownership range of a single processor, even though this can lead to an unbalanced data distribution when the number of processors does not divide the number of layers. Note that the subdomains used for subdomain deflation and the non-overlapping subdomains for the RAS preconditioner are exactly the same. This definition of the subdomains is used for the physics based data distribution.

6.1.1 Deflation

In a sequential environment, the deflation method has been shown to be very effective in combination with preconditioning. Of the considered preconditioners, the IC(0) preconditioner consistently provided the best results. However, setting up the preconditioner requires performing an incomplete LU-decomposition. This types of decomposition is notoriously inefficient to perform in parallel, as each processor requires information from many other processes to compute the decomposition. Therefore, the DJCG method is used for parallel computations, as the Jacobi preconditioner is simple to apply in parallel and does not require any communication between processors. The trade off is the increased number of iterations required to reach convergence.

CPU times and speedup for up to 7 processors are shown in Table 7. While the CPU time does decrease for a larger number of processors, this decrease is not linear. In fact, the times required for 4, 5, and 6 processors are almost exactly the same. However, the speedup when 7 processors are used is good at 6.39, and relatively close to the theoretical optimal speedup of 7. The fact that it is not closer to 7 can be explained by the extra communication costs, as well as a slight load imbalance introduced by the distribution of the data. The non-overlapping subdomains are used for distribution the data. This means that processors owning a sandstone layer have ownership of two more rows of discretisation points than processors owning a shale layer.

The observe CPU times can be explained by the data distribution as well. Each layer is owned entirely by one processor. However, the number of processors only divides 7 when it is either 1 or 7. For all other possible numbers of processors, the data distribution will introduce a large load imbalance. This cause of this imbalance is that some processors own more layers than others. In a parallel environment, the performance of the solver is decided by the slowest processor. In this case that is the processor with the most data. After each iteration, the other processors will complete their computations faster, and spend some time idle while waiting for the slowest processor to finish.

Processors	CPU time (s)	Speedup
1	27.280	1
2	18.190	1.50
3	11.609	2.35
4	7.941	3.44
5	8.041	3.39
6	7.891	3.46
7	4.269	6.39

Table 7: CPU times for the DJCG method with different numbers of processors

In Table 8, the maximum number of layers owned by a single processor is shown. Additionally, the theoretical maximum speedup is shown, assuming equal load for each layer. This explains why the CPU times are the same for 4, 5, and 6 processors, as in each case the processor with the largest workload has ownership of 2 layers. Considering the theoretical speedup, the achieved speedup is good for all cases. For 3 processors the achieved speedup is slightly larger than the theoretical speedup. This difference is small enough that it can largely be attributed to variance in the CPU times.

Processors	Max layers per processor	Theoretical speedup
1	7	1
2	4	1.75
3	3	2.33
4	2	3.5
5	2	3.5
6	2	3.5
7	1	7

Table 8: Maximum number of layers owned by a processor

6.1.2 Restricted additive Schwarz

The RAS method is known to be well suited to parallelisation. For a large number of processors, it is common that each processor has ownership over one subdomain. The correction on each subdomain can be computed without any need for communication between the processors. As observed before, the RAS preconditioner reduces the number of iterations required to reach convergence greatly. However, this does not necessarily result in faster times, as there are extra computational costs to computing the subdomain corrections. In parallel, the low number of iteration means that communication costs are very low.

Remember that the ICCG(0) method is used to compute the correction on every subdomain. This method is inefficient when performed in parallel. However, since each subdomain is stored on one processor, the correction on a given subdomain is computed on that one processor. Therefore, it is still possible to use this method, and it is not necessary to use another method such as the JCG method to compute the subdomain corrections.

In Table 9, CPU times and speedup are shown for different numbers of processors. The CPU times do increase for a larger number of processors. Similarly to the DJCG method, there is (nearly) no difference in CPU times for 4, 5 and 6 processors, due to the imbalanced data distribution. However, the speedup of 7 processors compared to 1 processor is only 3.66. This is far from the optimal speedup of 7. At first glance, this is unexpected, as there are very little communication costs involved with the method.

Processors	CPU time (s)	Speedup
1	78.260	1
2	49.455	1.58
3	45.195	1.73
4	25.477	3.07
5	25.408	3.08
6	25.418	3.08
7	21.380	3.66

Table 9: CPU times for the RASCG method with different numbers of processors

A closer look at the iterative solvers on the subdomains is required to find the cause of the low speedup. In Table 10, for every iteration, the maximum and minimum number of iterations of all the iterative solvers on the subdomains is shown. Additionally, the ratio between the maximum and minimum number of iterations is shown. For every application of the preconditioner after the first one, the processor with the highest number of iterations performs more than 5 times as many iterations than the processor with the lowest number of iterations. Assuming an equal number of discretisation points on each processor, this implies that the slowest processor takes 5 times as long to apply the preconditioner than the fastest processor. This leads to a large amount of idle time for the fast processors.

This explains why the speedup is much smaller than 7. The data points are (roughly) divided equally between the processors. For a CG iterations, this also means that the amount of work is (roughly) equally divided. However, for the RAS preconditioner this is not the case, and the amount of work for applying the preconditioner is not divided equally between processors. The subdomains that require a large number of iterations coincide with the sandstone layers between

shale layers. The subdomains that require a small number of iterations coincide with the shale layers. The top sandstone layer (with the Dirichlet boundary condition), requires a number of iterations roughly in the middle of the extreme values.

Iteration	Max subdomain iterations	Min subdomain iterations	Ratio
0	93	52	0.559
1	544	97	0.178
2	545	88	0.161
3	543	100	0.184
4	538	101	0.188
5	543	79	0.145
6	543	104	0.192
7	543	88	0.162
8	544	100	0.184
9	542	99	0.183
10	541	98	0.181

Table 10: Maximum and minimum number of iterations per subdomain at every iteration

6.1.3 Deflated RAS

Finally, the combination of deflation and RAS can also be parallelised. For this method, the relative tolerance is set to 10^{-14}, as otherwise the number of iterations remains too low to improve the error sufficiently. The method requires 8 iterations to reach convergence. The CPU times and speedup show roughly the same behaviour as for the RASCG method. This is as expected, since the subdomains are the same for both methods. Therefore, the same imbalance in the amount of work per processor holds for the application of the preconditioner.

Processors	CPU time (s)	Speedup
1	46.327	1
2	28.828	1.61
3	25.869	1.79
4	14.433	3.21
5	14.470	3.20
6	14.372	3.22
7	11.878	3.90

Table 11: CPU times for the DRASCG method with different numbers of processors

6.1.4 Accuracy of the solution

In Section 5.2.1, it is remarked that the true error does not improve by more than $c\kappa_2(A)u||\mathbf{x}-\mathbf{x}_0||_2$, where c is a small constant and $u \approx 10^{-16}$ is the (relative) machine precision. When the domain is divided into 100×100 elements, the improvement of the error is restricted to an order of 10^{-7}. It is well known that the condition number for a discretisation becomes larger when the grid is refined. For the parallel tests, the domain is divided into 1000×1000 elements, which is a much larger number of elements, leading to a much more refined grid. Therefore, the condition number of the discretisation matrix is larger. As a result, the improvement of the error is reduced to an order of 10^{-5}.

6.2 Increasing the number of processors

Up until this point, only cases where the number of processors is at most equal to the number of layers have been considered. In this section, the number of processors will be extended to exceed the number of layers. This requires some adjustment to the methods used, as it will no longer be possible for a subdomain to be within the ownership of a single processor. This will have different consequences for the deflation and RAS methods.

In the context of parallel computing, it is of interest to see if the deflation method and the RAS method work well in combination with the parallel data distribution. In large industrial applications, the number of processors will likely exceed the number of layers in a given problem. Ideally, the discretisation points are distributed evenly between the processors, this makes it most likely that the workload is the same for each processor. The most common configurations have the number of processors equal to some power of 2. Then the the computational domain is divided into a grid, using either, blocks, layers or stacks.

This is a different approach than has been used up to now. For example, for the tests with 7 processors a 1×7 configuration was used. This is somewhat unconventional, but in this case well suited to the problem.

The methods will be analysed for cases where the number of subdomains exceeds the number of layers. The aim is to have the subdomains coincide with a processors ownership range. This would mean that the application of either the deflation method or RAS preconditioner will require minimal communication between processors. This will however result in a change in either the deflated system or in the RAS preconditioner, as all results thus far have made use of the same definition of the subdomains. Therefore, some attention is given to the convergence of these methods with a larger number of subdomains.

For both methods, the definition of the subdomains is based on the position of the layers. This remains the same when the number of subdomains exceeds the number of layers. If (parts of) multiple layers fall within the ownership range of a single processor, a local subdomain will be created for every layer within the processors ownership. An example of this can be seen in Figure 30. The domain is divided among 4 processors in a 2×2 grid configuration. Each processor has ownership of the same number of discretisation points. For this particular example, each processor has (partial) ownership of 4 layers. Therefore, 16 subdomains will be created in total, 4 on each processor.

Note that even though the data is distributed evenly, the size of the subdomains is not necessarily the same. Furthermore, there are configurations for which the number of subdomains is different between processors. This is possible cause for an imbalanced workload.

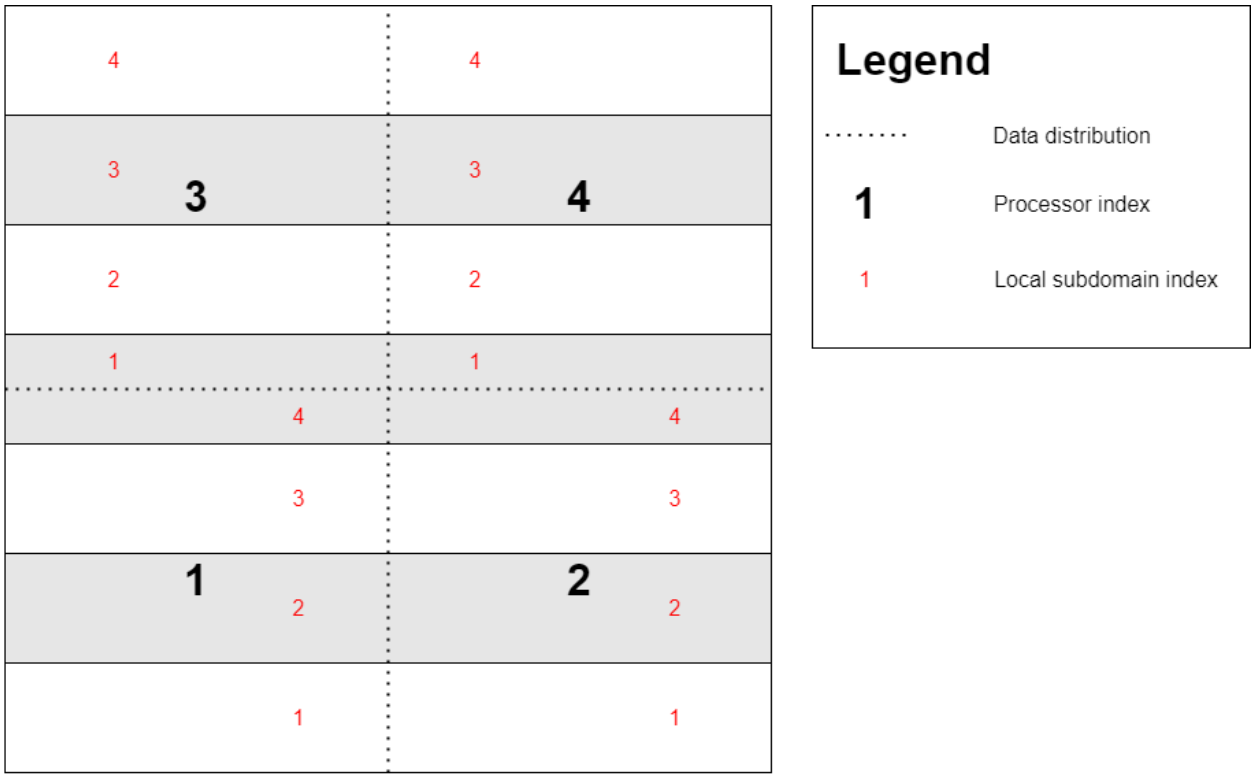


Figure 30: Example data distribution of model problem 2 with 4 processors arranged in a 2×2 grid

In total, three different types of subdomain layouts will be considered:

Equal data distribution
The data points are distributed evenly without taking the layers into account. Therefore, a single processor can have ownership of discretisation points in multiple layers. Then one subdomain is created for each layer that falls at least partially within a processors ownership range. This method has the benefit of using a very general data distribution, and application of the deflation or RAS method requires little communication.

Physics based data distribution
The layers are taken into account when distributing the data. Each processor only owns data points within a single layer. Therefore, only a single subdomain is owned by each processor. For this type of subdomain layout, the number of subdomains will be taken as a multiple of the number of layers. Any other number of subdomains would not result in a smaller workload for the worst case subdomain. This method requires a more specific data distribution based on the layers, but application of the deflation or RAS method requires little communication.

Shared subdomains
Only one subdomain is defined per layer. This is the same subdomain definition as used before. The subdomains are shared between processors. This method requires specific data distribution based on the layers. Additionally, the subdomains are shared between processors, requiring more communication for the application of either the deflation or RAS method.

Note that sharing the subdomains does not change the solution or convergence rate.

The goal of this section is to analyse the performance of these methods to see if they are viable in a parallel environment. For the purposes of this analysis, the testing will be done using the 100×100 element discretisation.

6.2.1 Parallel testing

Aside from the convergence analysis, it is also desirable to perform tests to determine the parallel performance of the deflation and RAS methods for the different types of data distributions. These parallel tests are performed on the Snellius cluster, on nodes of 128 processors. However, for all methods, a significant increase in CPU times is observed when the number of processors is larger than 8. This was first noticed when performing tests with 14 or 16 processors. However, upon closer inspection the increase in CPU time already happens when 9 processors are used.

For example, consider the DJCG method using equal data distribution for the 1×8 and 1×9 configurations. These methods are tested for the 1000×1000 discretisation, to ensure that the CPU times are large enough to be meaningful. The 1×8 configuration requires 4.61 seconds to reach convergence. The 1×9 configuration requires 231.08 seconds to reach convergence. The CPU time required to reach convergence is increased by a factor of 50. The communication costs, in contrast, only increase by a small amount. The 1×8 configuration has 7 interfaces between processors that require the communication of 1001 datapoints both ways at every iteration. The 1×9 configuration has 8 of these interfaces. Therefore the total communication costs should increase roughly by a factor of $1\frac{1}{7}$. This alone is not enough to explain the large increase in CPU time.

It is not entirely clear why there is such a large increase in CPU times. We present two possible explanations for this.

First, the Snellius cluster is a relatively new cluster. At the time of writing this report, the cluster has only been in use for a few months, and it is not unlikely that the system is not working correctly in some areas.

The second consideration is the shared memory within a node. Each node has 16 shared memory units, each connected to 8 processors. This means that runs with at most 8 processors can be performed using one single memory unit. Any configuration with more processors will require communication between memory units. It is possible that the bandwidth between the memory units is limited, and this is the cause of the increased CPU time.

Due to the large increase in CPU times for configurations with more than 8 processors, it is not possible to perform a meaningful comparison of the methods for more than 8 processors. However, it should not be taken as an absolute fact that the methods will not show speedup above 8 processors.

6.2.2 Deflation

For the deflation method the definition of the subdomains requires little change when there are more subdomains. The subdomains are defined by the boundaries of the layers and the boundaries of processor ownership. Discretisation points on the interface between two layers are assigned to the subdomain corresponding with the adjacent sandstone layer (i.e., the layer with the larger coefficient), similarly to the case of one subdomain per layer. The DJCG method is used here, as due to the parallel data distribution using the DICCG(0) method is inefficient.

Equal data distribution
For the equal data distribution, different grid configurations are taken into account for 4, 8 and 16 processors. Note that the case for 4 processors has fewer processors than subdomains. However, the data distribution is different than before, an leads to a different set of subdomains. Therefore, it is taken into account here. Results for these configurations can be seen in Table 12, both in terms of the number of iterations and the improvement of the error norm. Recall that the DJCG method converged in 220 iteration with a relative error norm of order 10^{-7} for the case with one subdomain per layer.

Grid configuration	#iterations	Relative error norm
1x4	202	1.60E-07
2x2	279	1.51E-02
4x1	267	7.73E-03
1x8	208	2.61E-06
2x4	283	5.47E-06
4x2	246	1.15E-02
8x1	304	1.51E-02
1x16	208	7.06E-07
2x8	297	2.72E-06
4x4	250	9.50E-06
8x2	275	1.29E-02
16x1	408	2.67E-02

Table 12: Number of iterations and relative error norm

The number of iterations varies between the different configurations. The method is most effective for configurations for which the data has been distributed in horizontal bars. For these configurations, the number of iterations is low (even slightly lower than for one subdomain per layer), and the error improvement is good. For the other configurations either the number of iterations is larger, or the error improvement is not sufficient.

Physics based data distribution
For the distribution by layers, the number of processors is always a multiple of 7. Results are shown for 7, 14, 21, and 28 processors. The layers are divided into smaller subdomains either horizontally or vertically. The results can be seen in Table 13.

Grid configuration	#iterations	Relative error norm
1x7	220	5.26E-07
2x7	206	2.41E-05
1x14	209	4.59E-07
3x7	247	6.43E-06
1x21	209	1.04E-06
4x7	250	1.09E-05
1x28	206	2.50E-06

Table 13: Number of iterations and relative error norm

There is not a very large difference between the different configurations. However, the configurations where the layers are divided into subdomains representing horizontal bars (2×7, 3×7, 4×7) are more consistent, requiring roughly the same number of iterations and obtaining a good improvement of the error. Dividing the layers into subdomains representing vertical bars results in a larger number of iterations, a worse error improvement or both.

Shared subdomains

For the case of shared subdomains, one subdomain is defined per layer. This is a situation that has been seen before, so the number of iterations and error improvement is known. However, the parallel implementation of the deflation method when the number of processors is larger than the number of subdomains causes some slight issues.

The restriction matrix Z has 7 columns. The PETSc package does not support sharing a matrix with 7 columns with more than 7 processors. Therefore, it is not possible to further investigate this situation for the purposes of this report. However, for a custom implementation it should be possible to perform parallel deflation when the number of processors is larger than the number of subdomains. The main problem in the PETSc package is the application of the restriction matrix Z and the computing the inverse of the coarse matrix $(Z^T AZ)^{-1}$. A custom routine can be written for this purpose. However, for the purposes of this report it is not possible to verify this due to time constraints.

Overall, the following conclusions can be drawn. When the data is distributed without regard to the layers, the deflation method only works consistently well when the subdomains are defined as horizontal bars across the entire domain. The number of iterations stays roughly the same regardless of the number of subdomains, and the error improvement is good.

When the layers are used to define the data distribution, the deflation method works most consistently well when the subdomains are defined as horizontal bars across the domain. Defining the subdomains as vertical bars generally shows either a worse convergence rate or a worse improvement of the error.

It is conjectured that it should also be possible to share the subdomains between processors. For the purpose of this report it was not possible to investigate this due to software limitations. However, the number of iterations and improvement of the error are known for this case, as the convergence behaviour for one subdomain per layer is known.

6.2.3 Restricted additive Schwarz

The RAS preconditioner requires a little more attention when defining the subdomains. Along the layer interfaces, the definition remains the same. For the subdomains without overlap, the points on the interface are assigned to the subdomain corresponding to the adjacent sandstone layer (i.e. the layer with the large coefficient). For the subdomains with overlap, these points are assigned to both adjacent subdomains.

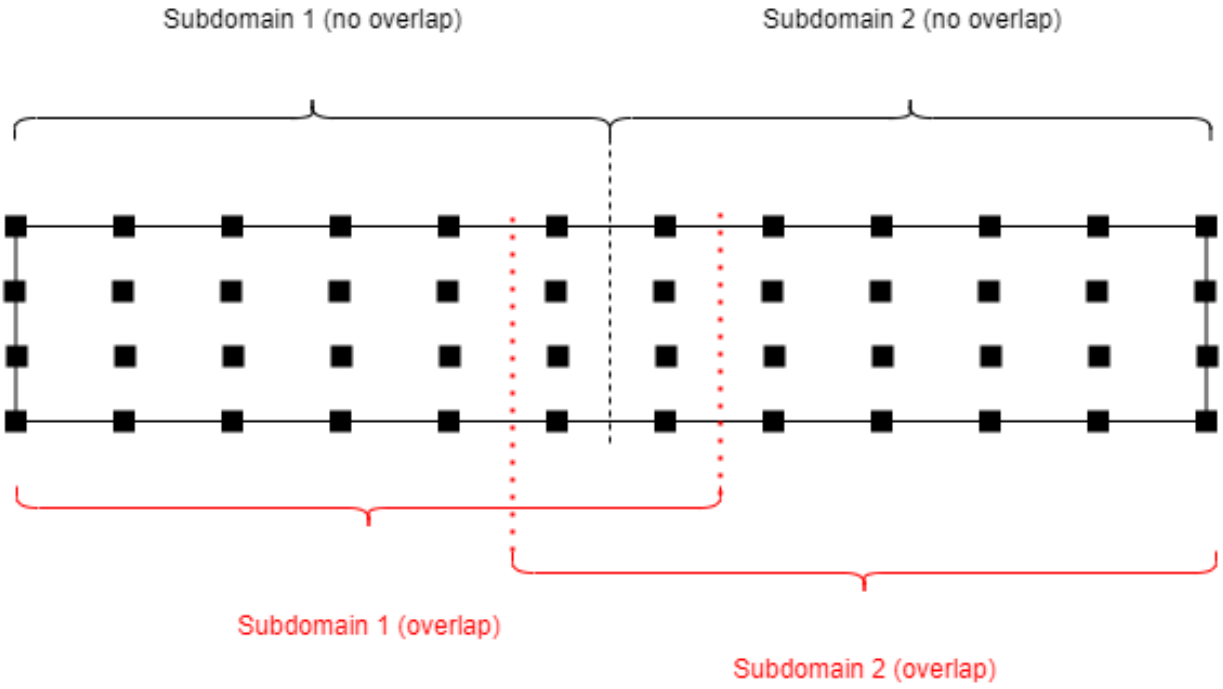


Figure 31: Example subdomain definition with and without overlap for two subdomains within one layer

A different scenario arises when two subdomains are adjacent within a layer. This situation occurs when the ownership of a layer is shared between processors. For both the equal data distribution and the distribution by layers this will result in two adjacent subdomains within the same layer. The subdomains without overlap are defined by the ownership of each processor within that layer. The subdomains with overlap are created by extending the non-overlapping subdomains by one row of points into the other subdomain. An example can be seen in Figure 31. In this example, two subdomains are defined within an arbitrary layer. Note that the number of discretisation points in the figure is not the same as in the 100×100 element discretisation.

Equal data distribution
For the equal data distribution, the different grid configurations are tested for 4, 8 and 16 processors again. The results can be seen in Table 14. Configurations for which no convergence was reached are indicated by a '-' mark. Recall that the RASCG method converged in 10 iterations with an error improvement of order 10^{-7} for the case with one subdomain per layer.

Grid configuration	#iterations	Relative error norm
1x4	30	2.97E-04
2x2	76	8.95E-05
4x1	123	2.73E-04
1x8	224	6.87E-05
2x4	79	1.19E-04
4x2	2034	1.15E-03
8x1	186	5.48E-04
1x16	312	4.99E-04
2x8	-	-
4x4	-	-
8x2	-	-
16x1	295	2.16E-04

Table 14: Number of iterations and relative error norm

The RASCG method shows largely different behaviour for each grid configuration. Some configurations do not reach convergence at all. All other configurations require significantly more iterations to reach a solution. This is especially worrisome as the work per iteration is large for this method. Additionally, the improvement of the error is worse by an order of 10^2 to 10^3.

Physics based data distribution
The result for the distribution by layers is shown in Table 15. Again, tests are done for 7, 14, 21 and 28 processors. The distinction is made between dividing the layers into subdomains in the shape of horizontal bars or vertical bars.

Grid configuration	#iterations	Relative error norm
1x7	10	1.23E-07
2x7	65	2.51E-05
1x14	-	-
3x7	102	2.33E-04
1x21	-	-
4x7	120	3.80E-04
1x28	-	-

Table 15: Number of iterations and relative error norm

Again, the behaviour of the method differs greatly for the different configuration. When the layers are divided into horizontal subdomains, the method does not converge at all. When the layers are divided into vertical subdomains, the method does converge, but the number of iterations is large, and the improvement of the error is worse by an order of 10^2 to 10^3.

Shared subdomains

For the RAS method, sharing the subdomains has several consequences. Firstly, the subdomain correction can no longer be computed with the ICCG(0) method, as this method is inefficient in a parallel environment. Therefore the JCG method is used, which requires more iterations to converge. Testing shows that the number of iterations for computing a subdomain correction is more than 3 times larger for the JCG method compared to the ICCG(0) method in the worst-case scenario.

Secondly, the computation of the correction on a subdomain requires communication between processors sharing a subdomain. The RAS method is appealing for the small amount of communication that is required in the application of the preconditioner. Since the number of iterations is low, the communication required for the CG iterations is also small. The number of iterations required for the computation of the subdomain corrections is much larger, and since the ICCG(0) method can not be used, the number of iterations will be even larger. This results in a very large increase in the communication costs.

As explained in Section 6.2.1, it is not possible to perform meaningful timing tests for more than 8 processors. However, it is possible to perform a small test on a slightly different problem. Consider a problem similar to model problem 2, but with 3 layers instead of 7. The top and bottom layer are sandstone, and the middle layer is shale. The coefficients are the same as in model problem 2. This problem is discretised using the finite element method with 1000×1000 elements. This problem is solved with the RASCG method using 3 and 6 processors. The method converges in 6 iterations. The corrections on the subdomains are computed using the JCG method. The results are shown in Table 16.

Processors	CPU time (s)	Speedup
3	50.31	1
6	27.11	1.86

Table 16: CPU time and speedup for 3 and 6 processors

The optimal speedup for 6 processors compared to 3 would be 2. The obtained speedup of 1.86 is not far removed from the optimal speedup. The difference can be explained by the extra communication costs required, mainly those needed when solving the subdomain corrections.

Overall, the RASCG method is not robust when the number of subdomains is increased. The number of iterations increases significantly, the improvement of the error is worse, and for many configurations the method does not converge at all. Due to the inconsistency it is not suitable to be used when the number of subdomains is larger than the number of layers.

It is not completely clear why the method works less well for these configurations. The increase in the number of iterations can partially be explained by the fact that the RAS method is known to have a slower convergence rate for a larger number of subdomains. Additionally, recall that the RAS preconditioner is asymmetric. For 7 subdomains the amount of asymmetry in the system is small, and the performance of the CG method is not affected significantly. A larger number of

subdomains increases the amount of asymmetry in the preconditioner. It is likely that this affects the effectiveness of the preconditioned CG method, as it is defined for a symmetric preconditioner.

It is possible to keep the number of subdomains low and share subdomains between processors. However, this results in larger communication costs. A small test shows that despite the extra communication costs, a good speedup can be obtained when the number of processors is doubled.

6.2.4 Deflated RAS

The DRASCG method uses the subdomain definitions provided for the deflation and RAS methods. Since it is not possible to perform any tests with shared subdomains for the deflation method, the shared subdomains section is omitted from this section.

Equal data distribution
For the equal data distribution, the different grid configurations are tested for 4, 8 and 16 processors again. The results can be seen in Table 17. Again, configurations for which no convergence was reached are indicated by a '-' mark. Recall that the RASCG method converged in 7 iterations with an error improvement of order 10^{-7} for the case with one subdomain per layer.

Grid configuration	#iterations	Relative error norm
1x4	13	5.30E-05
2x2	44	1.19E-02
4x1	74	2.08E-02
1x8	62	2.41E-06
2x4	25	8.24E-06
4x2	75	7.13E-03
8x1	62	4.12E-03
1x16	82	1.63E-07
2x8	-	-
4x4	40	4.13E-06
8x2	124	1.79E-03
16x1	-	-

Table 17: Number of iterations and relative error norm

Just like the RASCG method, the DRASCG method shows different behaviour for different configurations. In general, the number of iterations is much larger than for one subdomain per layer. Additionally, the error improvement is not good for most configurations. Some configurations do not reach convergence at all.

Physics based data distribution
The result for the distribution by layers is shown in Table 18. Again, tests are done for 7, 14, 21 and 28 processors. The distinction is made between dividing the layers into subdomains in the shape of horizontal bars or vertical bars.

Grid configuration	#iterations	Relative error norm
1x7	7	8.66E-07
2x7	24	1.67E-05
1x14	188	6.22E-07
3x7	30	6.31E-06
1x21	214	1.09E-06
4x7	34	4.46E-06
1x28	230	2.81E-06

Table 18: Number of iterations and relative error norm

Again, the DRASCG method performs less well when the number of subdomains is increased. The method does converge for all tested configurations, and the accuracy is generally good when compared to the RASCG method. However, for all other configurations, the number of iterations is very large when compared to the 1×7 configuration (one subdomain per layer).

The DRASCG method largely suffers from the same problems as the RASCG method. The number of iterations increases for configurations with more subdomains, often by a large margin. Just like the RASCG method, it is not robust for diffent configurations.

7 Conclusion

In this report, different methods for solving a layered problem with a large contrast in the coefficients are investigated. These kinds of problem arise in basin modeling when studying the evolution of a sedimentary basin. The large contrast leads to an ill-conditioned linear system, resulting in a poor convergence rate. For the purpose of this thesis, the research is restricted to a Poisson-like problem on a two-dimensional domain with 7 layers.

First, the behaviour of the CG method is investigated for a two-dimensional problem without layers, both without preconditioning, and in combination with the Jacobi and IC(0) preconditioners.

The same methods are used to solve the layered problem. The CG method is not able to efficiently solve the poorly conditioned system resulting from the large contrast in coefficients. The Jacobi and IC(0) preconditioners are able to improve the convergence rate significantly. However, the convergence rate is still not good, and not all problematic eigenvalues are removed from the system. As a result, the relative residual norm is not a reliable stopping criterion for these methods. If the relative tolerance is too large, the iterative method will terminate while the error is still large. Additionally, due to the large contrast in coefficients there is a limit to the improvement of the error.

Two main methods are investigated for this problem: subdomain deflation and restricted additive Schwarz preconditioning.

The deflation method is used to remove the effect of the remaining small eigenvalues. As these eigenvalues are associated with the jump in coefficients, subdomain deflation can be used to great effect, negating the need for potentially computationally expensive eigenvector approximations. The convergence rate of the CG method is greatly increased by using the deflation method in combination with preconditioning. Additionally, the relative residual norm can be used as a reliable stopping criterion.

The RAS preconditioner divides the domain into a set of overlapping subdomains and a set of non-overlapping subdomains based on the layers. At every iteration, a correction is computed on each subdomain using the CG method. Since every subdomain falls entirely within one layer, there is no contrast in coefficients within the subdomains. The RAS preconditioner allows information to travel through the domain much faster, and very few iterations are needed to reach convergence. However, the computation of the corrections on the subdomains adds computational costs. However, the relative residual is not a reliable stopping criterion for this method. The RAS preconditioner can be combined with subdomain deflation. The resulting DRASCG method requires very few iterations to reach convergence.

Comparison between these methods yields that the DRASCG method requires the fewest iterations to reach convergence. However, in a sequential environment, the DICCG(0) method needs the least time to reach convergence.

The methods are also investigated in a parallel environment. When the number of processors is smaller than the number of layers, the DJCG method requires the least CPU time and shows good speedup. The speedup for the RASCG and DRASCG methods is significantly worse, and CPU times are slower. The less than ideal speedup is caused by a load imbalance in the application of the preconditioner. In particular, the amount of iterations required to compute the correction on a subdomain differs greatly between subdomains.

In real-life applications, it is expected that the number of processors will exceed the number of layers. When dividing the data for such situation, it is generally best to perform a layer based data distribution. Distributing the data without regard for the layers leads to inconsistent behaviour of the solvers, mainly for the RASCG and DRASCG methods, which do not always reach convergence.

Distributing the layers into multiple subdomains works well for the DJCG method, especially

when the subdomains are defined horizontally across the domain. The RASCG and DRASCG methods work less well when the number of subdomains per layers is increased. A small test shows that the RASCG method has good speedup potential when the subdomains are shared between processors, despite extra communication costs.

To summarise, the DICCG(0) method is generally preferred as a method to solve problems with a large contrast in the coefficients. It is robust, even when the number of subdomains exceeds the number of layers. Additionally, it allows for the use of the relative residual as an effective stopping criterion. It outperforms the RASCG and DRASCG methods in timing tests sequentially. Similarly, the DJCG method outperforms the RASCG and DRASCG methods in parallel. Additionally, for a small number of processors the method has good speedup. The RASCG method, in contrast, does not allow the use of the relative residual as a reliable stopping criterion without combining it with the deflation method. The RASCG and DRASCG methods both require an extremely low number of iterations. However, the extra cost associated with computing the subdomain corrections causes it to lose out against the deflation method in timing tests both sequentially and in parallel. Additionally, the speedup is limited due to a large imbalance in the work load between different subdomains, even when the amount of data points is similar.

8 Further research

There are several avenues for further research, which fell outside the scope of this thesis due to time constraints.

In basin modeling, simulations involve a time-dependent problem with a growing domain due to the deposition of new sediments. It is hard to make optimal use of the parallel processing power due to an increasing number of discretisation points over the course of the simulation. The research in this thesis is restricted to a stationary and fixed domain. Future research could be used to investigate the effectiveness of the methods for a growing domain, especially in combination with the distribution and re-distribution of parallel data.

In this thesis, the RAS preconditioner is used in combination with the CG method, even though it is an asymmetric preconditioner. For a small number of subdomains the method works well, but for a larger number of subdomains the behaviour of the method becomes inconsistent. While the subject is touched on in this thesis, it is not investigated thoroughly enough to conclusively say why the method works for the given problem. Future research could focus on the general viability of the RASCG method.

More attention could be given to the parallel performance of the tested methods, especially when the number of processors is large. For the purposes of this thesis, it was not possible to completely investigate the parallel performance of the tested methods when the ownership of a subdomain is shared between processors. Therefore, it is unclear if it is better to increase the number of subdomains, or share the subdomains between processors, especially for the DJCG method.

References

[1] TB Jönsthövel, MB Van Gijzen, C Vuik, C Kasbergen, and A Scarpas. Preconditioned conjugate gradient method enhanced by deflation of rigid body modes applied to composite materials. *Computer Modeling in Engineering and Sciences (CMES)*, 47(2):97, 2009.

[2] C Vuik, A Segal, and JA Meijerink. An efficient preconditioned cg method for the solution of a class of layered problems with extreme contrasts in the coefficients. *Journal of Computational Physics*, 152(1):385–403, 1999.

[3] Xiao-Chuan Cai and Marcus Sarkis. A restricted additive schwarz preconditioner for general sparse linear systems. *Siam journal on scientific computing*, 21(2):792–797, 1999.

[4] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[5] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. PETSc/TAO users manual. Technical Report ANL-21/39 - Revision 3.16, Argonne National Laboratory, 2021.

[6] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil M. Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. PETSc Web page. `https://petsc.org/`, 2021.

[7] Thomas Hantschel and Armin I Kauerauf. *Fundamentals of basin and petroleum systems modeling*. Springer Science & Business Media, 2009.

[8] Xiaorong Luo and Guy Vasseur. Contributions of compaction and aquathermal pressuring to geopressure and the influence of environmental conditions. *AAPG bulletin*, 76(10):1550–1559, 1992.

[9] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[10] Kess Vuik and Domenico Lahaye. Lecture notes in scientific computing, September 2019.

[11] Roy A Nicolaides. Deflation of conjugate gradients with applications to boundary value problems. *SIAM Journal on Numerical Analysis*, 24(2):355–365, 1987.

[12] Zdeněk Dostál. Conjugate gradient method with preconditioning by projector. *International Journal of Computer Mathematics*, 23(3-4):315–323, 1988.

[13] Jason Frank and Cornelis Vuik. On the construction of deflation-based preconditioners. *SIAM Journal on Scientific Computing*, 23(2):442–462, 2001.

[14] Andreas Frommer and Daniel B Szyld. An algebraic convergence theory for restricted additive schwarz methods using weighted max norms. *SIAM journal on numerical analysis*, 39(2):463–479, 2001.

[15] Evridiki Efstathiou and Martin J Gander. Why restricted additive schwarz converges faster than additive schwarz. *BIT Numerical Mathematics*, 43(5):945–959, 2003.

[16] Henricus Bouwmeester, Andrew Dougherty, and Andrew V Knyazev. Nonsymmetric preconditioning for conjugate gradient and steepest descent methods. *Procedia Computer Science*, 51:276–285, 2015.

[17] Joost H van der Linden, Tom B Jönsthövel, Alexander A Lukyanov, and Cornelis Vuik. The parallel subdomain-levelset deflation method in reservoir simulation. *Journal of Computational Physics*, 304:340–358, 2016.

A Snellius cluster

National Supercomputer Snellius is used for timing tests. In particular, the so called CPU-only "thin" nodes are used. These nodes have the following specifications:
CPU SKU: AMD Rome 7H12 (2x), 64 Cores/Socket, 2.6GHz, 280W
CPU cores per node: 128
DDIM: 16x16GiB, 3200MHz, DDR4
Total memory per node (per core): 256 GiB, (2 GiB)
Other characteristics: 1xHDR100 ConnectX-6 single port, 2x25GbE SFP28 OCP