Comparison of Iterative Solvers for Non-Symmetric Linear Systems in Porous Media Problems

Master of Science Thesis in Applied Mathematics

Terje Haugen Lie

Department of Mathematics

University of Bergen

September, 2014

Acknowledgements

First and foremost I would like to thank my supervisors Jan Martin Nordbotten and Eirik Keilegavlen. I thank you both for giving me the opportunity to work under your supervision, it has been both cheerful and educational. Your knowledge and creative minds are highly motivational. I will be forever thankful for your assistance.

I thank my parents and siblings for pushing and motivating me to stay on track when times were rough. I thank you for taking an interest in my education, and for all your support.

I also thank all of my friends for making my days as a student this memorable. I will cherish these years for the rest of my life, and you are all a great part of it.

Finally, I would like dedicate all my hours of work at the University of Bergen to my late friend Alexander. He taught me a lot about the joys in life, but also made sure I had enough focus on my education. This one is for you, Captain!

Terje
August, 2014

List of content

Abstract

We investigate the performance of the IDR(s)-algorithms when solving nonsymmetric systems in porous media problems. We derive a mathematical model for the flow in porous media, and discretized this with the method known as Two-Point Flux-Approximation. By altering the permeability distribution and the grid size we design a series of systems with different sizes and properties. These systems have then been solved with IDR(s), using two different preconditioners, and the results have been compared against the popular GMRES.

We shall see that the short recurrence algorithm of IDR(s) appears in our cases to be an attractive alternative, compared to GMRES. Especially in the cases where both methods require a large number of iterations to solve the system shall we see the IDR(s) algorithm excel. However, we also encounter badly conditioned systems where the stability of the IDR(s) is not as good as GMRES.

This study is however not exhaustive, and more studies are need to identify under which conditions IDR(s) loses its stability. When this is identified, IDR(s) should after my consideration be considered as an attractive method for solving non-symmetric systems of linear equations in porous media problems.

Motivation

The need for solving linear systems with a large number of unknowns arises in many different fields; engineering, medical, environmental, etc. This thesis considers solving such systems, in the case where they describe the flow of some fluid in a porous media. Solving these large systems exactly is usually not attractive, due to the size of the problem. Instead, we search for good approximations to the solution with the use of an iterative method. Iterative methods all have in common that they generate a sequence of estimates that improves for each step. The idea is to start the sequence with an initial guess to the solution and by each step approximate a new and better solution. Once the approximation is sufficiently close to the true solution, it is taken as the solution to the system.

There exist a large number of iterative methods for solving large systems of linear equations. One of the classes of iterative methods for solving these systems is the Krylov subspace methods. These methods attempt to generate better approximations from the Krylov subspace. The choice of methods depends on the property on the system. For symmetric systems, the usual choice is the method of Conjugate Gradients. For nonsymmetric systems, the choice is not so clear. In general there is no superior method for solving nonsymmetric system. GMRES, proposed in 1986 by Saad and Shultz, is one of the most popular methods. However, it is quite expensive in terms of memory requirements. The other methods are not as robust, but are less expensive with respect to computational operations and memory requirements. The choice is therefore usually based on testing multiple methods on a specific problem.

The challenge lies in finding new and better algorithms for the iterative methods. The search for faster and more robust algorithms that require less computer memory and less CPU time is active field of research. In 2008 Sonneveld presented a new family of iterative methods, the IDR(s). IDR(s) was based on the nearly forgotten induced dimension reduction method. As we shall see further on in this thesis there is a clear relation between the methods, yet the ideas a completely different.

The aim of this thesis is to investigate the performance of the induced dimension reduction method, or IDR(s), presented by Sonneveld (2008) when solving systems that arise from discretizing a mathematical model for flow in porous media. To investigate the

performance of IDR(s) we will solve several different systems, and the results will be evaluated and related to one of the most popular and frequently used Krylov methods; the generalized minimal residual method (GMRES). Each system will be solved twice, with two different preconditioners, one algebraic and one geometric.

Chapter 1 is devoted to the mathematical model describing the flow of fluid in porous media. A discussion on the challenges of assembling the model is given, as well as important characteristics of the porous media and the equations describing the flow of the fluid. At the end of the chapter we end up with a complete model describing flow in porous media.

Chapter 2 focuses on the task of discretizing our model assembled in Chapter 1, and making it suitable for numerical evaluations. We derive the two-point flux-approximation scheme, based on the finite volume method and a Cartesian grid-system.

In **Chapter 3** we look at the concept of the condition number and how the idea of preconditioning affects this. A short discussion on the two preconditioners used in this thesis, ILU and ML, will be given.

In **Chapter 4** we introduce the iterative methods used in this thesis. We present their algorithms, and give a short discussion on the theoretical convergence and computational cost of the methods.

The presentation of the numerical results from our test cases will be given in **Chapter 5.**

Chapter 6 will be devoted to concluding remarks for the discussion given in the previous chapter.

Chapter 1

Challenges of modelling flow in porous media

Before we derive a complete model for describing the flow in porous media, we present the challenges that arise when assembling the equations that make up this model. Necessary properties and definitions of a porous media and fluids that flow in these will be given, and the challenges these present to our continuous model of flow will be discussed. We then introduce the equations that describe the flow in porous media, and the important characteristics of these. By the end of this chapter we will have a complete model flow of fluids in porous media.

For simplicity we will focus on single phase flow, which can be expanded for two of more fluids present in the porous media. Most of the following presentation is based upon [5]. For more details I refer to this book.

1.1 A Porous media

A porous media is a solid medium that contains pores. These pores can be described as "holes" or voids, and are randomly distributed throughout the media. For the fluids to flow through the media the pores have to be interconnected. These connected pores make up continuous pathways where the fluid can flow from one area of the material to another. These pathways in which the fluids flow are complex and complicated, and their length-scale are so small that they cannot easily be resolved, neither observationally of computationally. To describe these complex pathways mathematically we therefore introduce the notion of *representative elementary volume* (REV) [5]. The REV-method consists of giving a mathematical point the properties of a certain volume of material surrounding the point. The volume of REV should be large enough to give a representative average, but small enough to allow the properties to be approximated by continuous functions.

After introducing the notion of REV we can define the measurable property *porosity*, φ, of the rock. It is the scalar quantity that represents the volume of pores over the total volume within the REV,

$$\phi = \frac{volume\ of\ pores}{volume\ of\ REV}$$

From our approach of *representative elementary volume*, the actual porous media has been replaced with a fictitious continuum. In this fictitious media we have assigned an averaged value of porosity with respect to a surrounding volume to a mathematical point. The porosity is therefore assumed to be well defined and smooth. It can be differentiated or integrated, and is suitable for mathematical modeling [3].

1.2 The Darcy Law

One of the most important equations for describing flow of fluids through porous media is the Darcy law, named after Henry Darcy. He did several experiments on water treatment with different sand filters, and from this he made some observations that led him to predict how the water would flow through these filters. He found that the volumetric flow rate, q_d , could be written

$$q_d = K\frac{A(h_2 - h_1)}{l} \tag{1.2.1}$$

Where K is a proportionality coefficient, A is the cross-section area of the filter, $h_{1,2}$ are the respective hydraulic heads, and l is the length between the measuring points.

The hydraulic head is a measure of the pressure, p, (scaled by ρg) plus the elevation of the point, and is defined as $h = \frac{p}{\rho g} + z$. Where g is the gravitational acceleration, and z is the elevation of the measuring point relative to a set datum.

To express the volumetric flux, equation (1.2.1) is divided with A on both sides, and we get

$$u \equiv \frac{q_d}{A} = K\frac{(h_2 - h_1)}{l} \tag{1.2.2}$$

From Darcy's original linear equation we can now extend (1.2.2) into differential form. We let the fluid flow in three dimensions, assume the hydraulic head to be a sufficiently smooth function, and take the limit as l goes to zero. The Darcy law then takes the following differential form,

$$\boldsymbol{u} = -K\nabla h$$

One of Darcy's key observations was that the fluid flows from regions of higher hydraulic head to regions with lower. This explains the negative sign in front of the coefficient of proportionality, K, which we refer to as hydraulic conductivity.

1.2.1 Hydraulic conductivity and permeability

Hydraulic conductivity is an important property when describing flow in porous media. It is defined as

$$K = \frac{k\rho g}{\mu}$$

and is dependent on properties of both the fluid and the porous media. When a fluid flows in the interconnected pores it continuously deforms, and this deformation is referred to as *flow*. The fluid viscosity, μ, is the fluid's ability to resists this deformation. The fluid density, ρ, is simply defined as the mass of the fluid per unit volume of fluid. As for the effect of the property of the porous media on the hydraulic coefficient, permeability, k, is an average measure of the ability for fluid to flow through the porous media. Together with gravitational acceleration, g, these properties of the fluid and porous media define the hydraulic coefficient. In words, we can say the hydraulic conductivity indicates the ease of which a fluid flows through a porous media.

One of the challenging aspects of modelling flow in porous media is that the permeability of the medium may allow the fluid to flow more easily in one direction than another. When the hydraulic conductivity has no directional differences (isotropic), K is a constant scalar that indicates the ease of which the fluid flows through the porous media. Our volumetric flux vector, $\boldsymbol{u}$, is then related to the product of a constant scalar times the vector,

∇h. On the contrary, if the hydraulic conductivity is anisotropic we need to assign values to K that are dependent on the direction of flow. To preserve the vector form of our volumetric flux, $\boldsymbol{u}$, we need K to take the form of a conductivity matrix, $\boldsymbol{K}$. In this case, the Darcy law can be written

$$\boldsymbol{u} = -\boldsymbol{K}\nabla h$$

From the previous mentioned definition of the hydraulic head, we see that there are two driving forces in porous media: gravity and the pressure gradient. Substituting in our defined hydraulic head, we then write

$$\boldsymbol{u} = -\frac{\boldsymbol{k}}{\mu}(\nabla p + \rho \boldsymbol{G})$$

Where $\boldsymbol{G}$ is the gravitational acceleration vector. This form of the Darcy law is a more general statement of the relationship between fluid flow and the driving forces for that flow [5]. Since gravitational forces are approximately constant with a reservoir domain we can neglect the effect of gravity, and we only use pressure as the unknown.

$$\boldsymbol{u} = -\lambda \nabla p \qquad (1.2.3)$$

Where $\lambda = \frac{\boldsymbol{k}}{\mu}$.

It is worth stating that Darcy's law is only valid when the flow of fluid is so slow that the kinetic energy can be neglected (Laminar flow). When the fluid flows through the small complex pathways that define the porous media, the friction forces between the fluid and the pore walls will dominate and the flow tends to be very slow. Also, the influence of temperature and dissolved substances are neglected by assumption.

1.3 Mass Conservation Law

While the Darcy law describes how the fluid flows in a porous media, it is not sufficient for a complete model of the flow. To make it possible to obtain a unique solution to a general problem, a second equation has to be added to the model and we therefore look to mass conservation for a second equation. The mass conservation law is based on a simple and intuitive principle: any change of mass within a volume, Ω, must be a result of either mass flow through the boundary, $\partial\Omega$, or added mass to the volume that is not associated with the boundary. In mathematical terms we write

$$\int_{\Omega} \frac{\partial m}{\partial t} dV = -\oint_{\partial\Omega} f \cdot n \, ds + \int_{\Omega} r dV \qquad (1.3.1)$$

Where m is the mass per total volume of porous media, f is the mass flux vector, n is the outer normal to the surface $\partial\Omega$ and r is any sink or source terms within the volume. This equation states the above mentioned principle that the total change in mass over time (left side) is equal to the transfer of mass over the boundary (first term on right side) and the addition or removal of mass not associated with the boundary (second term on right side). To apply this equation to our model of flow, we express the variables above with respect to the known properties of the fluid and porous media. We write

$$m = \rho\phi,\ f = \rho\boldsymbol{u}, \text{ and } r = \psi \qquad (1.3.2)$$

where ψ represent sink of source terms of mass. Substituting the variables and with the use of Gauss theorem on the surface integral, the mass conservation can be written,

$$\int_{\Omega} \left(\frac{\partial(\rho\phi)}{\partial t} + \nabla \cdot (\rho\boldsymbol{u}) - \psi \right) = 0$$

Since the volume Ω is arbitrary and the integrand is assumed continuous we set the integrand itself to zero, and obtain the mass conservation law on a differential form, also known as the transport law,

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = \psi$$

For our simplified model, we assume the fluid to have constant density, and for the media to have constant porosity. The first term on the left side will then equal zero, and we arrive at our extension of the mass conservation law, which writes

$$\nabla \cdot \boldsymbol{u} = \frac{\psi}{\rho} \qquad 1.3.3$$

1.4 Complete model

Together, equation (1.2.3) and (1.3.4) now make up a complete model for single phase flow in porous media. It reads, for our case:

$$\nabla \cdot \boldsymbol{u} = \frac{\psi}{\rho}$$

$$\boldsymbol{u} = -\lambda \nabla p$$

These two equations can be combined with each other, and we arrive at the so called *pressure equation*

$$-\nabla \cdot \lambda \nabla p = \frac{\psi}{\rho} \qquad 1.4.1$$

Together with boundary conditions we now have a complete and closed model. A common practice in setting boundary conditions is to assume that no fluid can enter or exit through the boundary of the domain. Such a boundary condition is known as *no-flow boundary condition*, and usually the most common.

This model is based upon single phase flow under ideal conditions, and is therefore only valid under such conditions. However, this model can be generalized for two or more fluid by deriving one Darcy law and one transport law for each of the fluids present in the porous media. For such a case we need additional equations to close the system. For simplicity we have omitted the model for two- or multiphase flow. The latter discussion and mathematical operations are equally valid for the two- and multiphase, and can be done by extending the single phase model.

Chapter 2

Discretizing our model of flow in porous media

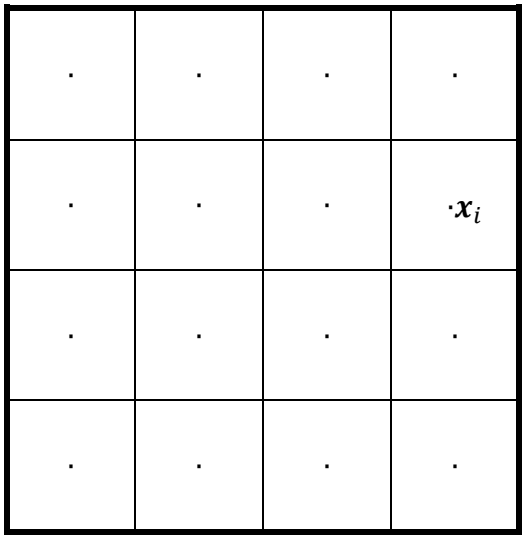
After dealing with the challenges of deriving a complete model for the flow in porous media we ended up with a complete model for describing the flow. This chapter will deal with the process of discretizing and making the continuous model suitable for numerical evaluations. Due to the memory capacity of the computer we cannot solve a continuous model numerically. Where an analytical solution solves our variables continuously throughout the domain, a numerical solution can only give approximate solutions to *discrete* points.

The following discretization will transform our continuous model into a sequence of discrete values for the domain. There are numerous ways of discretizing a domain, and in this thesis we are going to use a finite volume method of discretization. Our approach uses a Cartesian grid system with the gridlines aligned with the principal axes, and where each cell is represented by a cell centered average of the fluid pressure. This method is sometimes referred to as the *two-point flux-approximation* (TPFA) scheme [1].

2.1 Cartesian Grid system

Grids are generated by dividing our domain into smaller subdomains. We consider the situation where a domain Ω is divided into N subdomains Ω_i, and refer to each of the subdomains as *a cell*. We let the subdomains be non-overlapping, and the union of all subdomains combine to Ω. We refer to the edges between two neighboring cells i and j as a face, denoted $\partial\Omega_{i,j}$, and the cell center of Ω_i as $\boldsymbol{x}_i$. For a three dimensional problem these subdomains would represent volumes, where as in two dimension they would represent areas.

Figure 2.1 – A Cartesian grid system, with cell centers. Bold line around indicates the full domain, Ω, and the fine lines indicates the subdomains. All cell centers have been marked, and one has been labeled.



The Cartesian grid, which is used in this thesis, is a structured grid. The geometry of cells is arranged by letting the subdomains have faces that are aligned with the principal axes. This structure defines our grid, which we in the next will use to construct the finite volume method.

2.2 Finite Volume method

The finite volume method is family of numerical methods that discretely represents conservation laws [6]. It is based on the assumption that the conservation law holds for our domain, but also is valid for each cell in our discretized domain. As the name *finite volume method* implies, our domain is divided into a finite number of volumes. After dividing our domain into smaller subdomains, we return to the conservation law and generalize (1.3.1) for each subdomain. We then write

$$\int_{\omega_i} \frac{\partial \rho \phi}{\partial t} dV = - \oint_{\partial \omega_i} (\rho \boldsymbol{u}) \cdot n_{i,j} ds + \int_{\omega_i} \psi \, dV \qquad (2.2.1)$$

where the outward normal vector from one cell to the neighboring cells is denoted $n_{i,j}$. Using the same assumptions as in the previous chapter, the integral on the left hand side is still equal to zero. For the first term on the right hand side, we let the flux over the boundary from cell i

go to any neighboring cell j. By summing over all the faces to the neighboring cells j, the equation for mass conservation for each cell can then be written as

$$\sum_j \oint_{\partial\omega_{i,j}} (\rho \boldsymbol{u}) \cdot n_{i,j} ds = \int_{\omega_i} \psi \, dV \tag{2.2.2}$$

Keeping in mind that the density is assumed constant, and that the volumetric flux over a face can be and we write

$$\boldsymbol{u}_{i,j} = \oint_{\partial\omega_{i,j}} \boldsymbol{u} \cdot n_{i,j} ds \tag{2.2.3}$$

The conservation of mass in each cell then takes the following form

$$\sum_j \boldsymbol{u}_{i,j} = \int_{\omega_i} \frac{\psi}{\rho} dV \tag{2.2.4}$$

For this equation the flux is the unknown, and two solve this we use the Two-Point Flux-approximation to approximate the flux.

2.3 Two-Point Flux Approximation

The Two-Point Flux Approximation scheme is a discretization of the Darcy law. As the name states, it approximates the flux between two cells. In our case it uses the average pressure, assigned the cell centers, in two adjacent cells to approximate the flux over the face between the two cells. When this is done for all cells in our grid the scheme yields a complete mapping of the fluxes. This scheme was pioneered in [2].

The volumetric flux over a face, given by (2.2.3), and can be expanded using the Darcy law (1.2.5). The total flux from one cell over the faces between the adjacent cells is then given by

$$\boldsymbol{u}_{i,j} = -\int_{\partial\omega_{i,j}} \lambda \nabla p \cdot n \, ds \tag{2.3.1}$$

The pressure potential is assigned to the cell center, and to approximate the pressure gradient at the cell face we use central differences. The permeability is also defined as a cell wise constant, and is not defined at the edges. We must therefore also approximate λ on the faces between the cells. This can be done by taking a weighted harmonic average of the respective directional cell permeability.

After this has been done we end up with these approximations for the pressure gradient and λ.

$$\nabla p = \frac{2(p_j - p_i)}{(d_i + d_j)}$$

$$\lambda_{ij} = (d_i + d_j)\left(\frac{d_i}{\lambda_{i,ij}} + \frac{d_j}{\lambda_{j,ij}}\right)^{-1}$$

Where d_i and d_j are the respective distances from the face, $\partial\Omega_{i,j}$, to the cell centers. This can be used to rewrite (2.3.1), and the total flux over the face then takes the form

$$\boldsymbol{u}_{i,j} = 2\left|\partial\omega_{i,j}\right|(p_j - p_i)\left(\frac{d_i}{\lambda_{i,ij}} + \frac{d_j}{\lambda_{j,ij}}\right)^{-1} \tag{2.3.3}$$

To express the flux on a more compact form, and this is done by gathering the terms that do not involve the pressure into what is defined as *face transmissibility* $t_{i,j}$.

$$\boldsymbol{t}_{i,j} = 2\left|\partial\omega_{i,j}\right|\left(\frac{d_i}{\lambda_{i,ij}} + \frac{d_j}{\lambda_{j,ij}}\right)^{-1} \tag{2.3.4}$$

By summing the fluxes over all faces to adjacent cells, we get an approximation to the total flux over the faces, and we can rewrite (2.2.4)

$$\sum_j \boldsymbol{t}_{i,j}(p_i - p_j) = \int_{\omega_i} \frac{\psi}{\rho} dV \tag{2.3.4}$$

This can also be written as

$$\sum_j \boldsymbol{t_{i,j}}(p_i - p_j) = \boldsymbol{t}_{i,1}(p_i - p_1) + \ldots + \boldsymbol{t}_{i,i}(p_i - p_i) + \ldots +$$

$$\boldsymbol{t}_{i,j}(p_i - p_j) + \ldots + \boldsymbol{t}_{i,N}(p_i - p_N) = \frac{\psi}{\rho} \equiv \boldsymbol{q}$$

We have now reached a system of equations where the local fluxes can be explicitly represented as a combination of the pressure in adjacent cells. (2.3.4) is a linear system on the form

$$\boldsymbol{Ap} = \boldsymbol{q} \qquad (2.3.5)$$

In this thesis we use periodic boundary conditions, and therefore have to make this system positive definite. To do this we add a positive constant to the first diagonal of the matrix $\boldsymbol{A}$

Remark: This system is clearly a symmetric system, since the conservation law predicts flux to be the same magnitude, but with negative value, if we evaluate the flux from j to i instead. The goal of this thesis is however to investigating convergence behavior of IDR(s) when solving nonsymmetric systems. As we shall see in the latter, we will precondition our system using nonsymmetric preconditioners which will transform our system to being nonsymmetric and therefore suitable for our thesis.

Chapter 3

Preconditioning

After discretizing our mathematical model we ended the previous chapter with a system of linear equations. Before we look at the methods for solving this system, we will in this chapter briefly discuss the concept of condition number, preconditioning, and at the end we give a short description of the two preconditioners that will be used when solving our systems.

For this chapter, and the next, we consider a linear system on the same form as the one we derived in the previous chapter. We write it on a more general form, and for the remainder of this thesis we will refer to the non-symmetric systems in porous media problems as

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{3.1.1}$$

where $\boldsymbol{A} \in R^{N \times N}$, $\boldsymbol{b} \in R^{N}$ and $\boldsymbol{x}$ is unknown.

Before we present our preconditioner we shortly introduce the notion of condition number

3.1 Condition number

If a small perturbations of input data leads to a small change in output we say that the problem is well condition. On the contrary, we say that the problem is ill-conditioned if small perturbations of input data lead to large changes in output data. The meaning of "small" and "large" is related to the application. The condition number is used to measure the sensitivity of the solution to our system, with regards to small perturbations of input data. The problem of computing x, given b, has condition number

$$\kappa = \|\mathbf{A}^{-1}\| \frac{\|\mathbf{b}\|}{\|\mathbf{x}\|} \leq \|\mathbf{A}\| \|\mathbf{A}^{-1}\|$$

with respect to perturbations of $\mathbf{b}$ [12]. The product $\|\mathbf{A}\| \|\mathbf{A}^{-1}\|$ is the condition number of $\mathbf{A}$, denoted $\kappa(\mathbf{A})$:

$$\kappa(\mathrm{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\|$$

The condition number therefore only is attached to the coefficient matrix $\boldsymbol{A}$, and not the system. In the next chapter we will see that this is of practical important to us, since the convergence rate of iterative methods, such as GMRES and IDR(s), are indirectly linked to the convergence number. The convergence rate tends to decay as the condition number rises. To obtain the approximate solution to our system in a fast and accurate way, the condition number of our coefficient matrix is required to be as small as possible. If the condition number is not small enough, we can modify our system with the use of a *preconditioner*.

3.2 Preconditioning

The main idea of preconditioning is to design an effective matrix, the so-called preconditioner, in order to obtain a numerical solution with more accuracy or in less time [4]. If we consider system (3.1.1), the preconditioned system takes the following form

$$\boldsymbol{M^{-1}Ax = M^{-1}b} \tag{3.2.1}$$

The trick is to find some matrix $\boldsymbol{M}$, sufficiently close to $\boldsymbol{A}$, so that $\boldsymbol{M^{-1}A}$ has a better properties and condition number. This is based on the observation that for $\boldsymbol{M = A}$, we would have the ideal system $\boldsymbol{M^{-1}Ax = Ix = M^{-1}b}$ and all subspace methods would deliver the true solution in one single step [13]. The cost and time of constructing the preconditioner should be as low as possible. The bigger the difference between the costs saved by applying the preconditioner to the iterative method and the cost of constructing it, the more attractive the preconditioner is.

For this thesis, the choice has been made to use two different preconditioners. One is the well-known Incomplete Lower Upper (ILU) preconditioner, and the other is a Multi-Level preconditioner as described in [7]. A short description of these preconditioners will now follow.

3.3 Incomplete Lower Upper Preconditioner

The Incomplete Lower Upper (ILU) preconditioner has its name from Standard Gaussian elimination. Standard Gaussian elimination is the same as factoring the coefficient matrix, $\boldsymbol{A}$,

into a lower and an upper triangular matrix, respectively denoted $\boldsymbol{L}$ and $\boldsymbol{U}$. After $(n-1)$ steps of Gaussian elimination, we end up with and upper triangular matrix, $\boldsymbol{U}$, defined as:

$$\boldsymbol{U} \equiv \boldsymbol{A}^{(n-1)} = \boldsymbol{L}^{(n-1)}\boldsymbol{L}^{(n-2)} \dots \boldsymbol{L}^{(1)}$$

From this we define the lower matrix as the elimination matrix

$$\boldsymbol{L}^{-1} \equiv \boldsymbol{L}^{(n-1)}\boldsymbol{L}^{(n-2)} \dots \boldsymbol{L}^{(1)}$$

This leads to a factorization of our coefficient matrix, defined as

$$\boldsymbol{A} = \boldsymbol{LU}$$

The main problem in such factoring of a sparse matrix, as the once that arises from our discretizing techniques, is that the factors tends to be much less sparse than the original matrix A. In order to amend this problem, which causes the computation to be more expensive, the basic idea in the preconditioner is to set restrictions to the “fill in's” that occurs in the factorization. To preserve the sparsity of the system, the only non-zero entries in the factors $\boldsymbol{LU}$ are restricted to be the corresponding non-zero entries of $\boldsymbol{A}$. We then consider the factorization of $\boldsymbol{A}$ as

$$\boldsymbol{A} \approx \tilde{\boldsymbol{L}}\tilde{\boldsymbol{U}}$$

The incomplete factors of $\boldsymbol{A}$, $\tilde{\boldsymbol{L}}\tilde{\boldsymbol{U}}$, then defines the *Incomplete* Lower Upper preconditioner. For further use, we will only refer to this preconditioner as ILU.

Remark: ILU-preconditioners where proposed for positive and definite matrices with special structures. Due to this, it has been shown that this preconditioner has some difficulties providing robustness for a general matrix. Though this is the case, there has been a lot of theory developed proving that ILU-preconditioners are suitable for special classes of matrices, such as the once that arises from our discretization. Therefore, this is a suitable preconditioner for our system.

3.4 Multi-Level preconditioner

The other preconditioner we shall use in this thesis is Multi-Level preconditioner. This is a geometric multi-level preconditioner and is tailored to the give an approximation of the fine scale discretization on the coarse scale, and is based on conservation laws. A general description of the preconditioner will be given, but for details the reader is referred to [7]

To show how these preconditioners are constructed the notion of a 'coarse grid' needs to be explained. This can be done by recalling the grid system that was introduced in section 2.1, and defining this as the primal coarse grid. Each cell is then referred to as a primal coarse cell. These primal coarse cells consist of a set of interior cells from a finer grid, where the middle cell is defined as the 'vertex', see Figure 3.4.1.

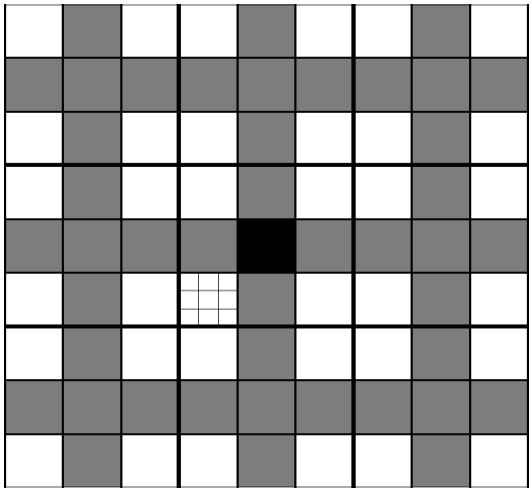


Figure 3.4.1: A Cartesian grid with fine and coarse (thick lines) cells; a finer grid is also indicated. The dual coarse grid is indicated by grey cells. The basis function centered in the black cell has support in all four surrounding dual coarse cells, and thus contribute to the flux expressions of all coarse edges shown in the figure. (Both figure and text are borrowed with curtesy from [7])

From the dual coarse grid shown in Figure 3.4.1, for each vertex a basis functions can be constructed with a one-dimensional version of the pressure equation (1.4.1). A matrix, ψ, can then be assembled by adding each basis function to its columns. Together with a matrix, φ, with piecewise constant test functions associated with the primal cells, the coarse discretization matrix A_c takes the form

$$\boldsymbol{A_c = \Phi A \psi}$$

This coarse linear system is then applied as our preconditioner.

Chapter 4

IDR(s) and GMRES

This chapter is devoted to our two iterative methods for solving non-symmetric systems. First, a general introduction to Krylov Subspace Methods is given. We thereafter present the mathematical ideas behind the IDR(s)-algorithm and look at the computational cost and theoretical convergence behavior, before we present GMRES and its properties. At the end of the chapter we make a short summary of the two methods and compare them to one another.

4.1 Krylov Subspace Methods

The Krylov subspace methods attempts to generate better approximations to the solution from what is known as the Krylov subspace, which is defined as

$$\mathcal{K}^k(\boldsymbol{A}; \boldsymbol{r}_o) = span(\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \boldsymbol{A}^2\boldsymbol{r}_0, \dots, \boldsymbol{A}^k\boldsymbol{r}_0)$$

where k is the iteration number, and $\boldsymbol{r}_0$ is the initial residual defined as $\boldsymbol{r}_0 = \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b}$, where $\boldsymbol{x}_0$ is the initial guess to our solution. We search for an approximate solution, $\boldsymbol{x}_k$, for which the residual $\boldsymbol{r}_k$ is within some desired tolerance. The residual, $\boldsymbol{r}_k$, can due to (4.1.1) also be given as a polynomial on the form

$$\boldsymbol{r}_k = \phi_k(\boldsymbol{A})\boldsymbol{r}_0 \qquad (4.1.1)$$

where ϕ_k is a k-th degree polynomial.

There exists a number of different Krylov Methods. One of the most popular methods is the generalized minimal residual approach, or GMRES. GMRES has the property of minimizing at every step the norm of residual vector over a Krylov Subspace [8]. As we shall see later, GMRES comes at the cost of having to compute and store new orthogonal basis vectors for the Krylov subspace at every iterations. If many iterations have to be performed in order to achieve the desired precision, the cost of memory and computations become prohibitive.

In the search for new and efficient Krylov methods, Sonneveld presented IDR(s) in 2008. This is a revised version of the *induced dimension reduction (IDR)* algorithm, presented by Sonneveld in 1980. As Sonneveld showed in his article describing the method, it has many desirable features. In the next section we present this method.

4.2 The IDR(s)-method

IDR(s) is a family of efficient, short recurrence methods for solving large nonsymmetric systems of linear equations. This algorithm, presented in [9], was based on a previous method introduced by Sonneveld in 1980, the Induced Dimension Reduction (IDR) method. IDR introduced a new way of solving these types of systems, as the underlying idea was completely different from the "usual" way of solving nonsymmetric systems. In contrast to many other methods, including GMRES, the IDR and IDR(s) methods generates residuals that are forced to be in subspaces of *decreasing* dimension. Though the IDR(s) method is built around this completely different idea, its features has clear relations to other Krylov-type solvers.

Before we look at the algorithm behind the iterative solver, we present the theorem that it is based on. The proof will not be given, for that I refer to [9] from where this section is based upon.

4.2.1 The IDR(s) theorem

As mentioned, IDR(s) is based on the IDR theorem from 1980. The theorem states

Theorem 4.2.1: The IDR theorem: Let $\boldsymbol{A}$ be any matrix in $\mathbb{C}^{NxN}$, let $\boldsymbol{u}_0$ be any nonzero vector in $\mathbb{C}^N$, and let $\mathcal{G}_0$ be the full Krylov subspace $\mathcal{K}^N(\boldsymbol{A}; \boldsymbol{u}_o)$. Let S denote any proper subspace of $\mathbb{C}^N$ such that S and $\mathcal{G}_0$ do not share a nontrivial invariant subspace of $\boldsymbol{A}$, and define the sequence $\mathcal{G}_j$, $j = 1, 2, ...$, as

$$\mathcal{G}_j = (\boldsymbol{I} - \omega_j \boldsymbol{A})(\mathcal{G}_{j-1} \cap S)$$

Where the $\omega_j's$ are nonzero scalars. Then the following hold:

(i) $\mathcal{G}_j \subset \mathcal{G}_{j-1} \; \forall \, j > 0$

(ii) $\mathcal{G}_j = \{\mathbf{0}\}$ for some $j \leq N$

According to the theorem it is possible to generate a sequence of nested subspaces of decreasing dimension, where the smallest possible subspace is $\{\mathbf{0}\}$. The IDR(s) algorithm, as we shall see next, consists of generating residual that are forced to be in these nested subspaces, $\mathcal{G}_j$. Applying this theorem to the algorithm then assures us that the problem will be solved after N steps of dimension reduction, at most.

What follows next is based on [9]. It is presented not intended as a practical, but instead a mathematical algorithm, and serves only as a justification for the algorithm used in this thesis.

4.2.2 The IDR(s) algorithm

At the end of Chapter 2 we ended up with a system of linear equations. This system was on the same form as (3.1.1), and for the rest of this chapter we will consider these types of systems. To solve the system (3.1.1) we start the algorithm with an initial guess, $\boldsymbol{x}_0$, to the solution to the system. This initial guess generates the initial residual, which is defined as $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$. For each step we look for a new and better approximate solution, $\boldsymbol{x}_k$, which again generates a residual defined as $\boldsymbol{r}_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k$.

A trivial observation is that if we are able to produce a recursion for the residual, $\boldsymbol{r}_k$, then we will also be able to produce a corresponding recursion for the approximate solution, $\boldsymbol{x}_k$. As mentioned in Section 4.1, the general Krylov Solver produces solutions for which the residual is forced to be the Krylov subspace. If the residuals and approximate solutions up to the k-th step has been calculated, then we will also be able to calculate $\boldsymbol{x}_{k+1}$ from

$$\boldsymbol{A}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = -(\boldsymbol{r}_{k+1} - \boldsymbol{r}_k) = [\phi_k(\boldsymbol{A}) - \phi_{k+1}(\boldsymbol{A})]\boldsymbol{r}_0$$

We can then express the general form of a Krylov-type solver as

$$\begin{aligned} \boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha\boldsymbol{A}\boldsymbol{u}_k - \sum_{l=1}^{\hat{l}} \gamma_l \Delta\boldsymbol{r}_{k-l} \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \alpha\boldsymbol{u}_k - \sum_{l=1}^{\hat{l}} \gamma_l \Delta\boldsymbol{x}_{k-l} \end{aligned} \tag{4.2.1}$$

where $\boldsymbol{u}_k$ is any computable vector in $\mathcal{K}^k(\boldsymbol{A};\boldsymbol{r}_0)\backslash\mathcal{K}^{k-1}(\boldsymbol{A};\boldsymbol{r}_0)$, $\Delta\boldsymbol{x}_k$ is the forward difference operator $\Delta\boldsymbol{x}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$, and $\hat{l}$ is the depth of the recurrsion.

After generalizing the Krylov-type solver we can now revisit the IDR-theorem. This theorem can be used to generate residual that are forced to be the subspaces, $\mathcal{G}_j$. The residual $\boldsymbol{r}_{k+1}$ will be in the subspace $\mathcal{G}_{j+1}$, if

$$\boldsymbol{r}_{k+1} = \left(\boldsymbol{I} - \omega_{j+1}\boldsymbol{A}\right)\boldsymbol{u}_k$$

where $\boldsymbol{u}_k \in \mathcal{G}_j \cap S$. With this restriction for the vector, $\boldsymbol{u}_k$, we choose

$$\boldsymbol{u}_k = \boldsymbol{r}_k - \sum_{l=1}^{s} \gamma_l \, \Delta\boldsymbol{r}_{k-l} \tag{4.2.2}$$

This leads us to (4.2.3) which describes the recursion of the IDR(s) family

$$\begin{aligned} \boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \omega_{j+1}\boldsymbol{A}\boldsymbol{u}_k - \sum_{l=1}^{s} \gamma_l \, \Delta\boldsymbol{r}_{k-l} \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \omega_{j+1}\boldsymbol{u}_k - \sum_{l=1}^{s} \gamma_l \Delta\, \boldsymbol{x}_{n-l} \end{aligned} \tag{4.2.3}$$

After deriving a recursion for the IDR(s) algorithm, we see that (4.2.3) is on the same form as the general Krylov-type solver given by (4.2.1).

In (4.2.3), s defines the depth of recursion. Since the usual choices of s is range from 1 to 16 (default value is 4), we have that $s \ll N$ and an algorithm that uses short recurrence, which is attractive with respect to computational and memory requirements.

The recursions for the solution and the residual are now defined, but are dependent on computing the coefficients γ_l. These coefficients can be computed in the following manner. We first assume that the space S to be the left null space of some $N \times s$ matrix $\boldsymbol{P}$ of full column rank, defined as

$$\boldsymbol{P} = (\boldsymbol{p}_1, \boldsymbol{p}_2, \dots, \boldsymbol{p}_s),\ S = \mathcal{N}(\boldsymbol{P}^H)$$

Note that s, known as the shadow vectors, is the codimension of the subspace S.

Since $\boldsymbol{u}_k$ is in $S = \mathcal{N}(\boldsymbol{P}^H)$, the relation $\boldsymbol{P}^H\boldsymbol{u}_k = 0$ holds. Combining this with (4.2.4), we end up with a $s \times s$ system for the coefficients γ_l, which has to be solved in order to determine $\boldsymbol{u}$ and $\boldsymbol{r}_{k+1}$. In [9] Sonneveld proposed to compute the residual $\boldsymbol{r}_{k+1}$ in the following way.

Define the matrices

$$\Delta\boldsymbol{R}_k = (\Delta\boldsymbol{r}_{k-1}, \Delta\boldsymbol{r}_{k-2}, \dots, \Delta\boldsymbol{r}_{k-s})$$

$$\Delta\boldsymbol{X}_k = (\Delta\boldsymbol{x}_{k-1}, \Delta\boldsymbol{x}_{k-2}, \dots, \Delta\boldsymbol{x}_{k-s})$$

After defining these matrices, the computation $\boldsymbol{r}_{k+1}$ is then carried out by the following algorithm

Calculate: $\boldsymbol{c} \in \mathbb{C}^s$ from the relation $(\boldsymbol{P}^H\Delta\boldsymbol{R}_k)\boldsymbol{c} = \boldsymbol{P}^H\boldsymbol{r}_k$

$$\boldsymbol{v} = \boldsymbol{r}_k - \Delta\boldsymbol{R}_k\boldsymbol{c}$$

$$\boldsymbol{r}_{k+1} = \boldsymbol{v} - \omega_{j+1}\boldsymbol{A}\boldsymbol{v}$$

When $s + 1$ residuals have been computed in $\mathcal{G}_{j+1}$ the next residual will be in $\mathcal{G}_{j+2}$. In the generic case, the decrease in dimension for the next subspace is then equal to s.

The scalar ω_{j+1} can be chosen freely when computing the first residual in $\mathcal{G}_{j+1}$, though the best choice is the value that minimizes the norm of the residual. This value must be kept the same during the calculations of all the $s + 1$ residuals in the same subspace. In the section about convergence, the choice of $\boldsymbol{P}$ is discussed. Once the residual is within the desired tolerance, we can update the approximate solution which yields our solution to the system.

4.2.3 Cost of IDR(s)

The cost of the IDR(s) algorithm is related to the choice of s. The computational cost and memory requirements increase for increasing s. To perform one full cycle of $s + 1$ IDR(s) steps, we can divide the cost into three parts. We need $s + 1$ matrix-vector products, $s^2 + s + 2$ inner products and $2s^2 + \frac{7}{2}s + \frac{5}{2}$ vector updates. This gives us that IDR(s) only needs one matrix-vector multiplication, which is the most costly operation, per iteration. The total cost for solving a system can then calculated from this.

Note that the cost per iteration is the same, regardless of which number is it.

4.2.4 Convergence behavior

Through what has been shown in the mathematical algorithm in section (4.2.2), after every $(s+1)$ iterations the residuals will be in a new subspace. This new subspace is of a lower dimension than the previous, and in most practical problems this dimension reduction has been shown to equal s. If this is the case, then in exact arithmetic it will require at most $N + \frac{N}{s}$ iterations to arrive at the exact solution.

As for the convergence rate, Sonneveld showed in [10] that the convergence behavior was dependent on two factors. By expressing the residual as the so-called residual polynomial, $\boldsymbol{r}_k = \phi_k(\boldsymbol{A})\boldsymbol{r}_0$, this can be rewritten as a product of two polynomials

$$\phi_k(\boldsymbol{A}) = \Omega_j(\boldsymbol{A})\psi_{k-1}(\boldsymbol{A})$$

Here, the $\Omega_j(\boldsymbol{A})$ are called the damping or stability factors, and $\psi_{k-1}(\boldsymbol{A})$ is called the Lanczos factor. These two factors have an independent influence the convergence rate of IDR(s).

The damping factors have their names because the factors ω_k, as mentioned earlier, are chosen minimize the norm of the residual that computed by the algorithm. The choice of ω_k's are therefore partially responsible for the convergence of the algorithm.

The Lanczos Factors is usually related to s, i.e. the rate of convergence will usually increase with increasing s. Though the value of s plays a role in the convergence behavior, it has also been shown that the choice of shadow vectors also plays a role. The best results are obtained if the vectors has as little to with the problem as possible.

As for the convergence rates, there is no way of telling the exact rate. This is due to the fact that no system exhibits the same properties, and this will lead to different rates for the convergence. What are shown above are factors that influence the convergence rate, but there is no way of explicitly telling how fast it will converge, other than practical examples.

4.3 The Generalized Minimal RESidual method

In this section we will cover the basic steps of the algorithm that builds up GMRES. This section is based upon [8] and [13], and for some of the terms used, the reader is referred to [12] for more information.

GMRES is a part of the class of Krylov methods that are known as the "minimal norm residual approach". In general we can say that the algorithm, at iteration step k, tries so identify the approximate solution $\boldsymbol{x}_k$ for which the norm of the residual, $\|\boldsymbol{b} - A\boldsymbol{x}_k\|$, is minimal over the Krylov subspace $\mathcal{K}^k$.

4.3.1 Finding a suitable basis

Due to the fact that for each iterations the vectors $\boldsymbol{A}^k\boldsymbol{r}_0$ points more and more in the direction of the dominant eigenvectors causes the basis $\{\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \boldsymbol{A}^2\boldsymbol{r}_0, \dots, \boldsymbol{A}^k\boldsymbol{r}_0\}$ to be an unattractive basis for the Krylov subspace. Before we start computing the approximate solution we need a suitable basis. This is done by the Arnoldi process, which orthonormalizes the basis for the Krylov subspace. There are many ways of doing this, but the most used is the modified Gram Schmidt procedure.

The process of making a suitable basis starts with normalizing the initial residual, which then is defined as $\boldsymbol{v}_1 = \frac{\boldsymbol{r}_0}{\|\boldsymbol{r}_0\|}$. Using this normalized residual we can now compute $\boldsymbol{A}\boldsymbol{v}_1$. This is then orthogonalized with respect to $\boldsymbol{v}_1$ and the result is again normalized, which yields $\boldsymbol{v}_2$. This process is repeated for each step to create the basis for the Krylov subspace. Given that we already have an orthonormal basis $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_k$ for the Krylov subspace $\mathcal{K}^k(\boldsymbol{A}; \boldsymbol{v}_1)$, this basis is expanded by computing $\boldsymbol{v}_{k+1} = \boldsymbol{A}\boldsymbol{v}_k$ and orthonormalizing this vector with respect to the basis.

From the Arnoldi process we also initiate a k by $k-1$ upper Hessenberg matrix, $\widehat{\boldsymbol{H}}_{k,}$, with entries defined by the process. For more information on the Hessenberg matrix, see [12]. This matrix is in direct relation to coefficient matrix, $\boldsymbol{A}$. If the last row of $\widehat{\boldsymbol{H}}_{k,}$ is removed, then relation between the two would be a similarity transformation. The eigenvalues of the coefficient matrix will then be preserved.

From this, the following relations are true

$$\boldsymbol{A}\boldsymbol{V}_k = \boldsymbol{V}_{k+1}\widehat{\boldsymbol{H}}_k \tag{4.3.1}$$

where $\boldsymbol{V}_k$ is the matrix with columns $\boldsymbol{v}_1$ to $\boldsymbol{v}_k$.

4.3.2 Minimizing the residual norm

The last step of the GMRES algorithm is the step that minimizes the residual norm over the Krylov subspace, and then gives the approximate solution. We then look for an approximate solution on the form $\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{z}$, for which the residual norm over $\boldsymbol{z}$ in $\mathcal{K}^k$ is minimal. This gives us a least square problem to solve, represented by

$$min_{\boldsymbol{z}\in\mathcal{K}^k}\|\boldsymbol{b} - \boldsymbol{A}[\boldsymbol{x}_0 + \boldsymbol{z}]\| = min_{\boldsymbol{z}\in\mathcal{K}^k}\|\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{z}\| \tag{4.3.2}$$

The vector $\boldsymbol{z}$ can be represented as $\boldsymbol{z} = \boldsymbol{V}_k\boldsymbol{y}$, where $\boldsymbol{y}$ is a k-dimensional vector. We define $\beta \equiv \|\boldsymbol{r}_0\|$ and the function $f(y) = \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_i\|$ and use the above stated relation to rewrite (4.3.2)

$$f(y) = \|\beta\boldsymbol{v}_1 - \boldsymbol{A}\boldsymbol{V}_k\boldsymbol{y}\| \tag{4.3.3}$$

Using the relation stated (4.3.1) we obtain

$$f(y) = \left\|\beta\boldsymbol{v}_1 - \boldsymbol{V}_{k+1}\widehat{\boldsymbol{H}}_k\boldsymbol{y}\right\|$$

which can be factorized into

$$f(y) = \left\|\boldsymbol{V}_{k+1}\left[\beta\boldsymbol{e}_1 - \widehat{\boldsymbol{H}}_k\boldsymbol{y}\right]\right\|$$

Where $\boldsymbol{e}_1$ is the first column of the identity matrix. Remember that $\boldsymbol{V}_{k+1}$ is composed of vectors that are orthonormal to each other, and that the norm such matrices are one, this gives us that

$$f(y) = \left\|\beta\boldsymbol{e}_1 - \widehat{\boldsymbol{H}}_k\boldsymbol{y}\right\| \tag{4.3.4}$$

The solution to the least square problem is then given by

$$\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{V}_k \boldsymbol{y}_k \tag{4.3.5}$$

Where $\boldsymbol{y}_k$ minimizes the function (4.3.4) over $\boldsymbol{y} \in R^k$

The final norm can then be minimized by solving the minimum norm least square problem for the k by $k-1$ upper Hessenberg matrix, $\widehat{\boldsymbol{H}}_k$, and the right hand side $\beta \boldsymbol{e}_1$. This result can then be given as

$$\widehat{\boldsymbol{H}}_k \boldsymbol{y} = \beta \boldsymbol{e}_1$$

This least square problem is then solved by realizing that $\widehat{\boldsymbol{H}}_k$ has a QR decomposition. For more information on the QR decomposition see [12]. Due to the upper Hessenberg structure of the $\widehat{\boldsymbol{H}}_k$ matrix, this can efficiently be done with plane rotations, also known as Givens rotation. The Givens rotation annihilates the sub diagonal elements of $\widehat{\boldsymbol{H}}_k$. This results in a k by $k-1$ upper triangular matrix which is denoted $\boldsymbol{R}_k$, whose last row is zero. This results in the relation

$$\boldsymbol{Q}_k \widehat{\boldsymbol{H}}_k = \boldsymbol{R}_k$$

Where $\boldsymbol{Q}_k$ is a k by k matrix and the product of the successive Givens eliminations of the sub diagonal elements of $\widehat{\boldsymbol{H}}_k$. After this transformation y the minimizes the least square problem which can be rewritten as

$$\begin{aligned} f(y) &= \left\| \widehat{\boldsymbol{H}}_k \boldsymbol{y} - \beta \boldsymbol{e}_1 \right\| = \| \boldsymbol{Q}_k^T \boldsymbol{R}_k \boldsymbol{y} - \beta \boldsymbol{e}_1 \| \\ &= \| \boldsymbol{R}_k \boldsymbol{y} - \boldsymbol{Q}_k \beta \boldsymbol{e}_1 \| \end{aligned}$$

This leads us to the final least square problem to the minimum norm solution

$$y = \boldsymbol{R}_k^{-1} \boldsymbol{Q}_k \beta \boldsymbol{e}_1$$

Which leads us to our approximate solution, $\boldsymbol{x}_k$, given as

$$\boldsymbol{x}_k = \boldsymbol{V}_k \boldsymbol{y}$$

Note that from minimizing function the residual norm is nothing but $f(y)$ which is equal to $\|\boldsymbol{R}_k\boldsymbol{y} - \boldsymbol{Q}_k\beta\boldsymbol{e}_1\|$. By the construction of y, this norm is the absolute value of the last component of $\boldsymbol{Q}_k\beta\boldsymbol{e}_1$. The residual norm of the approximate solution, $\boldsymbol{x}_k$, is therefore calculated at the end of each iterate. This prevents us from having to specifically calculate the residual norm at each step. The algorithm will therefore keep iterating until the residual norm is within the desired tolerance, and we do not have to calculate the approximate solution at every step.

To summarize the algorithm, we can easily generalize the algorithm into four important steps:

1. With the Arnoldi process, generate a suitable basis for the subspace, and initiate the upper Hessenberg matrix.
2. Minimize the norm of the residual, $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k\|$
3. Repeat until the residual is within the desired tolerance
4. Compute the approximate solution $\boldsymbol{x}_k$

4.3.3 GMRES(m): A restarted version

One of the most obvious challenges with GMRES is the cost of storing and calculating the basis for the Krylov subspace. As k increases, so does the number of vectors requiring storage. To remedy this challenge, there exist a version of GMRES which is known as GMRES(m), or *restarted GMRES*. In this version, the algorithm is restarted at every m step, where m is some fixed integrer much smaller than N. If the residual norm is not within the desired tolerance after m iterations we set $\boldsymbol{x}_0 = \boldsymbol{x}_m$ and restart the algorithm, which allows us to clear the storage.

There is no rule for determining a suitable m. Restarting at some m may work better for some matrices than other. Since the speed of convergence may vary drastically for different m, even for those close to one another. The only way to determine a suitable m is by trial and error.

4.3.4 Cost of GMRES

The cost of computing the approximate solution, $\boldsymbol{x}_k$, is for GMRES increasing for each iteration. As we have seen, a new basis for the Krylov subspace has to be computed and stored for each iteration. For a low number of iterations this is not a problem, but as the number of iteration grows, this becomes quite significant. For each iteration the algorithm performs one matrix-vector multiplication. To account for the other computational operations we divide the GMRES algorithm into two parts, the Arnoldi process and the formation of the solution, we can easier account the number. The Arnoldi process requires approximately $k(k+1)N + kF$ multiplications, and the formation of our solution kN. The total cost of calculating our solution comes at $k(k+2)N + kF$ multiplications, where F denotes the number of nonzero entries in our coefficient matrix $\boldsymbol{A}$. In addition to this, as mentioned above, the computation also requires storage of vectors. As k increases the number of vectors to be stored increases like k, which can become quite prohibiting as k grows.

For the restarted algorithm, GMRES(m), the total cost is $k\left(m + 3 + \frac{1}{m}\right)N + kF$. But we only have to store the orthonormalize vectors, the $\boldsymbol{v}_i$'s, the approximate solution $\boldsymbol{x}_k$, and the vector for $\boldsymbol{A}\boldsymbol{v}_i$. The storage is then reduced to $(m+2)N$

4.3.5 Convergence Behavior

In exact arithmetic GMRES will converge to the true solution within the N-th step. At the N-th step the Krylov subspace spans $\mathbb{R}^N$ and therefore the minimizing step will find the exact solution, but as mentioned before, this is very inefficient for large systems since the storage increases drastically as k grows. On a more general basis, we say that the convergence of the algorithm is monotonic. This is true due to the fact that the residual is minimized over the Krylov subspace, and since $\mathcal{K}^{k+1} \supset \mathcal{K}^k$ the minimization over a larger subspace will yield a smaller residual, or at worst a residual equal to that of the previous subspace.

Though it is easy to show that the convergence of the algorithm is monotonic, the e rate of convergence is more difficult to predict. To establish some useful insight we can use the fact that the nonsymmetric nature of the coefficient matrix $\boldsymbol{A}$ may have eigenvalues that are complex. These eigenvalues appear in the complex plane, and are within N circle, and can be bounded by an ellipse. The size of this ellipse can be used to determine the convergence

rate of GMRES , i.e the closer the eigenvalues are to one another, the faster GMRES converges.

4.4 A short summary

The obvious difference between the two methods is the subspace from which they generate the residual. IDR(s) starts out with the full Krylov subspace and there after uses a series of nested subspace of decreasing dimension, where GMRES uses a subspace of increasing dimension. We have also seen that GMRES has a standard algorithm without free parameters; after choosing an initial guess the algorithm computes the solution by minimizing the residual at each step. IDR(s) has three free parameters, s, ω_j, and $\boldsymbol{P}$, that can impact efficiency of the algorithm. The choice of the shadow space, s, affects the dimension reduction per full cycle, and therefore the total number iterations. $\boldsymbol{P}$ defines the subspace S, and ω_j is chosen to minimize the residual. The differences mentioned above are the most obvious differences between the two, but as we also have seen the IDR(s) algorithm can be expressed as a general Krylov solver and there are therefore many common features between the two methods.

We also say that GMRES is the optimal method with regards to iterations. In exact arithmetic GMRES solves the system in N iterations, whereas IDR(s) solves it in $N + \frac{N}{s}$ iterations. For each iteration, both of the methods preform one matrix-vector multiplication. This matrix-vector multiplication is the most costly of all the computational operations, but the same for both methods. The big difference between the two is that the memory requirement and computational operation for IDR(s) is constant, independent of the number of iterations. On the contrary, GMRES needs to compute and store a new basis for the Krylov Subspace for each iteration. This becomes quite prohibiting as the number of iteration.

Chapter 5

Numerical Results

To investigate how IDR(s) performs when solving our discretized model of flow in porous media, a series of test cases will now follow. All of the test cases have been done with MATLAB R2014A programed on a standard stationary computer. The performance of IDR(s) will be evaluated and discussed, and the results are compared to GMRES. These test cases are all examples of systems of linear equations that arise from discretizing the model of flow in porous media.

To test IDR(s) on different systems, a series of linear systems have been constructed. These systems have been assembled accordingly to the first two chapters, and their properties have been altered by changing the permeability distribution. Six different permeabilities have been tested. One of each one listed below.

- Homogeneous
- Randomly distributed heterogeneous permeability
- Randomly distributed binary heterogeneous permeability
- Log-normal permeability
- The SPE10 dataset. This dataset consist of 85 layer, where the top 35 layers have a relatively smooth permeability and the bottom 50 layers have well defined long channels with high permeability. The methods will be tested on two layers. One from the top, layer 27, and one from the bottom, layer 51.

In addition to changing the permeability, the discretized has been done with 5 different grid sizes. The finest grid size we have used is $\frac{1}{288}$ in each direction. The other 4 grids are the same structure, but doubles in size at each step. The biggest grid size we will use is then $\frac{1}{18}$. For each permeability the methods will then be tested on five different linear systems, with N ranging from 324 to 82 944.

For each case we have solved the system with two versions of preconditioned IDR(s) and GMRES each, using the two preconditioners described in the Chapter 3. When solving with IDR(s), we have chosen four values of s; 1, 2, 4, and 8. The default choice of P and ω has

been used, with $P = randn(N, s)$ and $\omega = 0.7$. IDR(s) has be implemented in MATLAB with Sonneveld's own algorithm, and GMRES with a special memory saving code. Instead of full GMRES, a version called "reduced memory" has been used. This is not the restarted version, but a version where the matrix used to store the representation of the Krylov subspace is not preallocated. In this case, the size of the matrix grows dynamically instead of be the full size from the start.

The focus of our investigation that follows will be on the convergence behavior of IDR(s) on these types of systems. Important features are the total number of iterations, the memory usage to reach the approximate solution within the desired tolerance. The tolerance is set to $1{,}00 \times 10^{-7}$in all cases. As an indication on how the computational operations and memory requirements are balanced in each case, the run time for the total process of computing the solution will also be given. The only change made in the GMRES algorithm is that what has mentioned, we therefore include its run time for comparison. Together with the iteration count, run time can give us a good indication of the performance of IDR(s) compared to GMRES, and conclusions can be drawn from this.

The results are divided into three sections. First we look at the cases where our methods where preconditioned with ILU. Not all of the results from every case are given, but the general behavior will be explained and a representative example will follow. In the cases where some of the IDR(s)-versions were not able to compute a solution within the specified tolerance, the test cases will be highlighted and discussed. Secondly, we repeat the above mentioned for the cases where we used the Multi-Level preconditioner. Concluding remarks will be given in the next chapter.

5.1 Preconditioned with ILU

As mentioned above, this section will discuss the cases where the methods were preconditioned with ILU. We first focus on the cases where the desired solution was computed and section 5.1.1 will generalize these results. In section 5.1.2, the cases where IDR(s) did not compute the desired solution will be covered. Finally, in section 5.1.3, we make some remarks and make a short summary of the previous two sections.

5.1.1 General behavior of IDR(s)

In general, solving larger systems needs more iterations than smaller systems, and as the permeability gets rougher and exhibits large variations within close range the iteration number goes further up. The total number of iterations is also dependent on the value of s. As the theory in chapter 4 predicted, the total number of needed iterations is decreasing for an increasing value of s. When we compare IDR(s) and GMRES, IDR(8) uses between 10 and 30 percent more iterations than GMRES. Except for this general behavior the results from each test case was fairly similar. We will therefore generalize the result, and this will be discussed with a representative example. This example will together with some extra comments be representative for all, only excluding cases that will be covered in section 5.1.2.

The representative example is from a case where the media has a homogenous permeability distribution. We have discretized our continuous model according to Chapter 2, using square grids with length $\frac{1}{288}$, and end up with a linear system of 82 944 equations. We precondition our system with the ILU-preconditioner and the system is solved with IDR(s) and GMRES. When solving this system with the two iterative methods the residual can be plotted against the number of iterations, and we end up with a convergence plot, see figure 5.1.1.

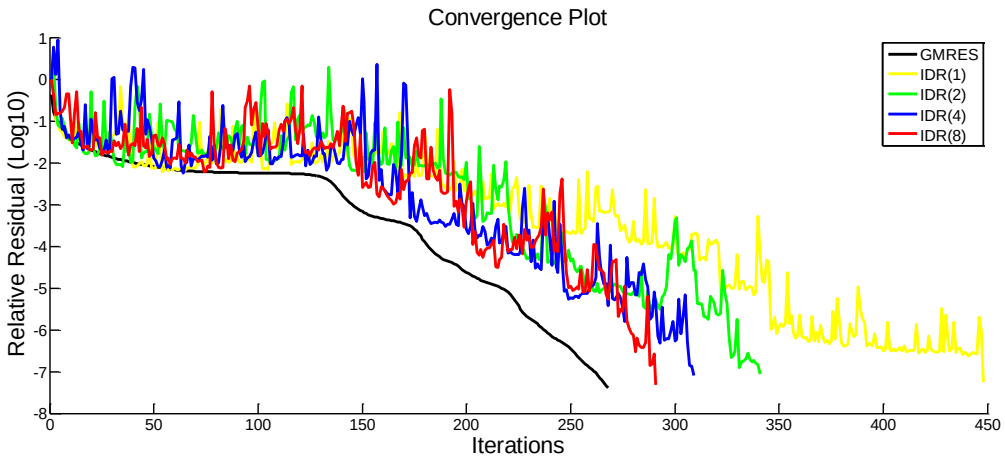


Figure 5.1.1 - Convergence plot of our representative example

As we can see from figure 5.1.1 the curve of IDR(8) is fairly close to the curve of GMRES. We clearly see that the convergence of IDR(s) is dependent on the value of s, and converges faster when s increase. For a precise number of iterations and run time needed to compute the desired solution by each method see table 5.1.1.

Table 5.1.1 – Number of iterations and elapsed time for our representative example

Method	Number of iterations	Elapsed time (s)
GMRES	267	122.71
IDR(1)	447	5.23
IDR(2)	340	4.49
IDR(4)	308	5.37
IDR(8)	290	6.90

In this specific example, the difference in number of iterations between IDR(8) and GMRES is only 23. Table 5.1.1 shows us the number of iterations performed by each method. Even though the number of iterations goes down for increasing values of s, the computation operations and memory requirements of IDR(s) increases as s increases. This cost is as we have seen constant for each iteration. GMRES, on the other hand, has to store and compute a new orthogonal basis for every iteration. This becomes quite costly when the iteration number increases. To get an indication on the cost of the two methods, table 5.1.1 also presents the run time for reaching the approximate solution for each method.

As we can see, GMRES uses approximately twenty times more run time than what is used by IDR(2). This is a major difference, and reflects how attractive a short recurrence method like IDR(s) is when N is large. Also notice that IDR(2) only uses 65 percent of the time that IDR(8) uses on the same problem. Though the difference between the two versions is 50 iterations, the number of inner products and vector updates per iteration goes up, as well as the memory requirement, when s increases. The decrease in iterations is in this case not large enough, compared to the increase in computational operations.

In this specific case IDR(2) was the fastest of the tested methods, which indicates that the combination of number of iterations, computational operations, and memory requirement was well balanced. This was not only the case for this example, but also for more than 50 percent

of the test cases. Even though the total number of iterations need to compute a solution goes down when s increases, both memory and computational requirements goes up per iteration.

An important feature of IDR(s) is that when N gets bigger and bigger, the convergence curves of IDR(4) and IDR(8) looks more and more like GMRES. However, the cost of GMRES become more expensive with respect to memory requirements as the iteration count goes up. The low memory requirements of IDR(s) compared to GMRES causes IDR(s) to use significantly less time to reach a solution. This difference in run time therefore grows drastically as N gets bigger, and for some our biggest systems the fastest IDR(s) versions only used approximately 3 percent of what GMRES used to solve the same system.

5.1.2 Stagnation of IDR(s)

In this section we take a closer look at the cases where some of the IDR(s)-versions were not able to compute a desired solution. This occurred in three cases, one with IDR(4) and two with IDR(8). All of these three cases happened when using permeability fields from the SPE10 dataset, and only when we used the finer grid systems.

In a case where we used Layer 27 and grid size $\frac{1}{288}$, both IDR(4) and IDR(8) returned a solution that was not within the tolerance. Table 5.1.2 shows total number of iterations, run time, and relative residual for all methods. Remember that the tolerance was set to 1.00e-7.

Figure 5.1.2 – Total number of iterations, run time, and relative residual for case with stagnation

Method	Number of iterations	Elapsed time (s)	Relative Residual
GMRES	728	850.63	9.8397e-8
IDR(1)	2455	26.89	9.2222e-8
IDR(2)	1456	18.39	6.4038e-8
IDR(4)	857	13.83	1.3947e-7
IDR(8)	956	24.08	1.2834e-7

When IDR(4) and IDR(8) returned the residual, it also returned a so called "Flag 2" indicating that this is the lowest residual that this algorithm possibly can produce. There could be some ways of working around this, and in [11] some steps for remedying this problem is discussed. In this thesis this has not been done, so for more information on the remedy I refer to this manual. However, a notice should be made about possible explanations. The condition number of our coefficient matrix gets higher as the grid gets finer and the contrast in permeability gets larger. We also know that the number of computational operation grows with increasing s, which again increases the round-off errors in MATLAB. Since the stagnation happened with IDR(4) and IDR(8), this could be an explanation. This is strengthen by the fact that when considering layer 51 described in SPE10, and grid size $\frac{1}{144}$, IDR(8) stagnated with a relative residual equal to 2.4761e-7, after 670 iterations.

In these cases when IDR(s) do not return the desired solution, we see in Table 5.1.2 that due to the high number of iterations, the run time for IDR(4) and IDR(8) is significantly lower than the that of GMRES. However, IDR(1) and IDR(2) compute the solution and their run time is also significantly lower than GMRES. In these cases, where our system is badly conditioned, IDR(s) seems to perform better when s is low.

5.1.3 Summary

In all cases GMRES solved the systems with fewest iterations, and IDR(8) was the runner up. This result in not very surprising, and reflects the theory discussed in chapter 5. GMRES is theoretical optimal with respect to iterations, since it in exact arithmetic uses at most N iterations to compute the exact solution. In exact arithmetic IDR(8) solves the system in $N + \frac{N}{8}$ total iterations, which is 12.5 percent more than GMRES. In our results this was in some cases as low as 10 percent, but sometimes as high as 30.

Though GMRES solves the system with the least iterations, it comes with a cost of storing and computing a new orthonormal basis for the Krylov subspace at each step. This becomes prohibiting as the iteration count grows. The cost of IDR(s) is constant for each iteration. Because of this, all IDR(s)-versions use significantly less time to solve the system than GMRES when the iterations count is high. In our results we have also seen that IDR(s) is faster than GMRES when the iteration number is low, but this difference is not as large.

When solving some of the larger systems, arising from the different layers in SPE10, we saw IDR(4) and IDR(8) stagnate at a relative residual larger than what we set as our tolerance. In these cases we expect the system to be badly conditioned, and they therefore required a large number of computational operations. Since IDR(4) and IDR(8) were the only versions to stagnate above the tolerance, it is plausible that this could be caused by a build-up of round-off errors.

This gives us an indication that the ILU preconditioned IDR(s) can be a good alternative when solving systems that arise from discretizing equations that describes flow in porous media. To prevent stagnation of the residual s should be kept small (1 or 2) for larger problems where the permeability field exhibits large variations. In the cases where IDR(s) did stagnate the algorithm returned the lowest possible residual within 10 percent of the time used by GMRES to compute the solution, and the loss of time is therefore not severe. The possibility to save this amount of run time makes IDR(s) a good alternative.

5.2 Preconditioned with Multi Level preconditioner

We now look at the cases where the Multi-Level preconditioner was used. The structure of this section will be the same as the previous section.

The performance of IDR(s) is looks to be more dependent on the performance of the preconditioner. In general we can say that when the Multi-Level preconditioner successfully approximates the geometry of the permeability on the coarse scale, IDR(s) works better than when an algebraic preconditioner is used. On the other hand, when the Multi-Level preconditioner is used on the binary permeability distribution and the lower levels of the dataset SPE10, it is not predicted that the preconditioner is quite as successful. In these cases we will that IDR(s) performs less efficiently than when preconditioned with ILU, compared to GMRES.

This section will cover the results in the following way. First we consider the cases where the preconditioner worked well, and solving the system showed a clear system that stems from this. Second we look into the cases where the preconditioner is not as successful, and finally we look closer at the systems where a desired solution was not computed, and shortly discuss the possible explanations for this.

5.2.1 A successful Multi-Level preconditioner

This section will cover the cases with homogenous and lognormal permeability distribution and Layer 27 from the SPE10 dataset. These permeability fields have in common that the Multi- Level preconditioner successfully approximate the coarse scale geometric structures, and that this not dependent on the size of the coarse grid system. In these cases the iteration number is expected to stay unchanged even for increasing N, both IDR(s) and GMRES. These three permeability distributions will be covered by on example, which is representative for all.

The representative example is the cases where layer 27 in SPE10 was considered. To get a good picture on how this preconditioner influences the performance of our methods, we will in this example cover all of the grid sizes. After solving the systems, Table 5.2.1 shows the total number of iterations need to solve the different systems.

Table 5.2.1 – Number of iterations for all grid sizes for solving problems with Layer 27 in the SPE10 dataset

Grid size	Iterations				
	GMRES	IDR(1)	IDR(2)	IDR(4)	IDR(8)
$\frac{1}{18}$	5	8	9	8	7
$\frac{1}{36}$	6	8	8	8	7
$\frac{1}{72}$	6	10	9	9	9
$\frac{1}{144}$	7	13	11	11	11
$\frac{1}{288}$	6	10	10	9	9

As the table shows, the number of iterations needed to compute the solution stays nearly unchanged even though the number of equations doubles for each system. Since this preconditioner in these cases is successful, it causes the number of iterations to stay nearly unchanged, regardless of grid size.

When the preconditioner worked this well, and this pattern occurred, all of the IDR(s)-versions obtained the solution faster than GMRES. However, since the number of iterations is low the difference in run time not large.

5.2.2 A less successful Multi-Level preconditioner

In the cases where the permeability was more challenging, such as random and binary distribution and layer 51 in SPE10, we encountered a situation where the preconditioner is not expected to be successful. In these cases the convergence behavior of both IDR(s) and GMRES resembled what we saw with the ILU preconditioner. For increasing N, the number of needed iterations to compute the solution increased. For increasing s we saw the number of iterations used by IDR(s) drop, and GMRES still used less iterations than all IDR(s) versions. The total number of iterations needed to compute the solution for each system was lower was still lower than when ILU was used, for the same systems. However, the number of iterations used by IDR(s) was no longer as close to GMRES.

Again we look at a representative example for these results. This specific example a domain with random distribution of heterogeneous permeability, discretized with grid size $\frac{1}{144}$. After solving the system, Table 5.2.2 shows the total number of iterations and the total run time for each method.

Table 5.2.2 – Total number of iterations and run time for the example with a less successful preconditioner

Method	Iterations	Elapsed Time (s)
GMRES	21	0.78
IDR(1)	47	1.10
IDR(2)	40	0.93
IDR(4)	34	0.82
IDR(8)	30	0.78

From the table 5.2.2 we see that GMRES uses the same amount of time as the fastest IDR(s)-version, which in this case is IDR(8). It also shows that the total number of iterations used is relatively low, and the difference between GMRES an IDR(s) is relatively large. As we mentioned in the earlier cases, GMRES gets increasingly more costly as the number of iterations grows. The first iterations are therefore ‘cheaper’ than the later, so when GMRES uses few iterations and IDR(s) uses that many iterations more they come out at the same run

time. A difference in the general behavior from what we saw when we used ILU preconditioner was that the number of iterations for IDR(4) and IDR(8) no longer got as close to GMRES. In all cases, IDR(8) used approximately 50 percent more iterations than GMRES used to solve the same system.

A general behavior of IDR(s), compared to GMRES, is that as the number of iterations rises, the run time of IDR(s) becomes more attractive due to the short recurrence algorithm. However, in these cases we saw the IDR(s) versions use less time than GMRES for the smaller systems (grid size $\frac{1}{72}$ and up), and GMRES being the fastest for the bigger systems.

It should be noted that at some point IDR(s) is predicted to be faster than GMRES again. When the cost of computing and storing the basis for the Krylov subspace get large enough, IDR(s) will again be faster than GMRES.

5.2.3 Cases with stagnation

In addition to the behavior we have covered in the two sections above we also encountered some cases where some IDR(s)-versions did not compute a desired solution. When IDR(s) did not compute the solution this was caused by the so-called Flag 2, as we saw in section 5.1.3. In addition to this we had one case of IDR(1) using the max number of iterations which was preset to N^2. In one of the cases we also experienced a stagnation of GMRES before the desired residual was met. These cases all happened when we solved the bigger systems that described flow in binary permeability distribution and layer 51 in the dataset SPE10.

We will cover the case where both IDR(s) and GMRES stagnated, and therefore only considered the binary permeability distribution and the finest grids, $\frac{1}{72}$ and $\frac{1}{144}$. The comments made about IDR(4) and IDR(8) in this example also applies for the SPE10 case that we will not discuss.

In the case of binary permeability distribution and the finer grids, we encountered stagnation for both IDR(4) and IDR(8). In addition to this stagnation IDR(1) used the maximum number of iterations when solving the system arising from the finest grid. Table 5.2.3 shows the iteration count and Table 5.2.4 shows the relative residual. A red number indicates stagnation, and green indicates max iterations.

Table 5.2.3 – Total iteration count for solving system describing binary permeability distribution

N	Iteration				
	GMRES	IDR(1)	IDR(2)	IDR(4)	IDR(8)
5 184	10	145	116	89	110
20 736	33	20736	3086	733	364

Table 5.2.4 – The relative residual returned by the methods after stagnation, termination, or computing solution

N	Relative residual				
	GMRES	IDR(1)	IDR(2)	IDR(4)	IDR(8)
5 184	8.4763e-8	5.5083e-8	6.9895e-8	6.497e-5	0.00033351
20 736	1.1206e-7	0.0076216	6.1429e-8	0.00022213	4.5388e-5

Notice that when $N = 20\ 736$ the only method that is able to solve the system with the desired tolerance is IDR(2). However, it used 3086 iterations which is not very impressive. For the same system, GMRES also stagnated, and returned a notice that the condition number of the matrix times the unit round-off, machine precision, was greater than the tolerance. However, GMRES returned a relative residual that was only 12 percent over the tolerance. This is much less than the residual that IDR(4) and IDR(8) were able to compute.

From the results in our cases it seems like the robustness of IDR(s) goes down when s increases, and especially when the preconditioner is not working optimally. This is most likely caused by round-off error, since the number of computational operations goes up when s increases, but as GMRES notified the condition number of the coefficient matrix is high. This should therefore be weighted carefully.

In the cases where we encountered stagnation IDR(s) proved to be most stable and robust when s is low. In the case where GMRES stagnated and IDR(2) was able to obtain a solution, GMRES used 33 iterations before it stagnated. IDR(2) used 3086 iterations to compute the desired solution. Table 5.2.5 shows the run time for the cases discussed above, and as we can see IDR(2) used more than 50 times the time used by GMRES.

Table 5.2.5 – Total run time for each method before stagnating, terminating or computing the solution

N	Elapsed time (s)				
	GMRES	IDR(1)	IDR(2)	IDR(4)	IDR(8)
5 184	0.20	1.15	0.89	0.69	0.91
20 736	1.45	533.92	80.92	19.61	10.40

In both of the cases where IDR(s) stagnated it used more time than what GMRES did. In the case where $N = 5\ 184$, the iteration number for IDR(s) is still relatively low, but when $N = 20\ 736$ this number goes drastically up. This causes IDR(s) to use considerably more time to solve the system, and when GMRES still uses a low number of iterations this cases a large difference in run time. In these cases GMRES is clearly the most attractive method. Even in the case where GMRES stagnated, the low number of iterations and the fact the residual was only 12 percent off causes it to be an attractive method.

5.2.4 Summary

From the results of our test cases, we see that the performance of our methods is dependent on the performance of the Multi-Level preconditioner. When the preconditioner works well, the convergence of IDR(s) is close to the optimal GMRES independent on the choice of s. IDR(s) solves in all cases the system in less time than GMRES does. In the case where the permeability field is not as smooth and exhibits large differences within close range, the preconditioner is not as successful. In these cases the iteration number for the IDR(s) version is not as close to the GMRES as they were when preconditioning with ILU. Since the iteration number in these cases is still relatively low compared to N in these cases, and the difference between IDR(s) and the GMRES relatively large, IDR(s) is no longer the obvious choice.

We have in this section also seen some cases where IDR(s) stagnates before returning the desired solution. In one of these cases, GMRES also stagnated, and returned a notice that the condition number was bad. In these cases, as in the previous section, we have seen that IDR(s) preforms best in these cases when s low. In the next chapter a summary of both sections will be given. Some concluding remarks based on the result from our cases and some thoughts on IDR(s) will be given.

Chapter 6

Concluding Remarks

In this thesis we have investigated the convergence behavior of the IDR(s) family of solvers when using it to solve systems that arise from discretizing a mathematical model of flow in porous media. Through a series of test cases where the properties of the systems of linear equations have been altered, and with the use of two different preconditioners, we have encountered a number of different results which leads us to draw these concluding remarks.

In general, the number of iterations IDR(s) has to perform to compute the desired solution goes up when the number of unknowns goes up. In addition to this, we have also seen the number of iterations increase further when the permeability becomes rougher and exhibits large scale differences within close range. This is not surprising and reflects on the fact that the condition number grows when the permeability becomes rougher and the grid size gets smaller, and since the convergence rate of IDR(s) is indirectly linked to the condition number we see this. We have also seen that the number of iterations needed to compute the desire solution with IDR(s) is dependent on the chosen value of s. In exact arithmetic IDR(s) solves the system in $N + \frac{N}{s}$ steps, so a decrease in number of iterations is expected for larger values of s. This has been confirmed in all cases.

We have chosen to compare the performance of IDR(s) with the well-known GMRES. GMRES is optimal with respect to the number of iterations, since it in exact arithmetic solves the system in N iterations at most. Throughout our test cases with ILU as preconditioner we saw the difference in the iteration numbers between IDR(8) and GMRES vary from 10 to 30 percent. In these cases all IDR(s)-versions solved the system faster than GMRES. As the number of unknowns went up, the difference in run time increased. This algebraic preconditioner is assumed to work equally well on all systems, and we therefore saw this system of behavior in all test cases. When a large number of iterations have to be performed by both methods, IDR(s) is considerably faster than GMRES. This makes IDR(s) especially attractive when solving larger systems of systems that require many iterations before a desired solution is computed, compared to GMRES. This is due to the fact that the computational cost and memory requirements are constant for IDR(s) in each iteration, whereas these properties grow exponentially for each iteration with GMRES.

When preconditioning with the Multi-Level-preconditioner, we do not expect the preconditioner to work well for all permeability distributions. We know that when the permeability distribution is rough and exhibits large scale differences within close range, this preconditioner is not optimal. However, since the preconditioner has several attractive features we chose to apply this to our methods for all permeability distributions. In the cases where the permeability was smooth, and the Multi-Level-preconditioner worked well, both IDR(s) and GMRES used a low number of iterations, regardless of grid size and permeability. The IDR(s)-versions was still the fastest one, but the difference between the two methods was no longer as significant as in the cases where we used the ILU-preconditioner. However, in our test cases where the permeability distribution was of such a character that the Multi-Level-preconditioner no longer was optimal we saw that the IDR(s)-versions struggled more than what GMRES did. In these cases we saw the IDR(s) versions use more iterations than in the cases where the preconditioner work well, relative to GMRES. This caused IDR(s) to no longer be the fastest method in these cases.

We have also seen some cases where IDR(s) stagnates and therefore not compute the desired solution. This stagnation happened when using the finer grids on rougher permeability fields, which in general leads to badly conditioned systems. In these cases we saw IDR(s) stagnate for the higher values of s (4 and 8), whereas it performed better when s was low (1 and 2). We have also seen through our cases that IDR(s) is more dependent of the performance of the preconditioner than GMRES, which should be taken into account

Compared with GMRES we have seen that IDR(s) is a very attractive method when solving systems that requires many iterations. The short recurrence algorithm requires a constant amount of computational operations and low memory requirements per iteration, which is attractive when the number of iterations goes up.

The study in this thesis is however not exhaustive, and more studies are need to identify under which conditions IDR(s) contains its stability. If this is identified, IDR(s) should after my consideration, be considered as an attractive method for solving non-symmetric systems of linear equations in porous media problems. This is based on the opportunity of saving significantly amounts of time used to compute the solution to these problems.

Bibliography

[1] J.E. Aarnes, T. Gimse, and K.-A. Lie. *An Introduction to the Numerics of Flow in Porous Media using Matlab.* In G. Hasle, K.-A. Lie, and E. Quak (Ed.), Geometrical Modeling, Numerical Simulation, and Optimization: Industrial Mathematics at SINTEF. Springer Verlag, 2007.

[2] K. Aziz and A. Settari. *Petroleum Reservoir Simulation.* Applied Science Publishers, 1979.

[3] J. Bear. *Dynamics of fluids in porous media.* Dover Publications, 1988

[4] K. Chen. *Matrix Preconditioning Techniques and Applications.* Cambridge University Press, 2005.

[5] J. M. Nordbotten and M. A. Celia. *Geological storage of CO_2: Modeling approaches for Large-Scale Simulations*. John Wiley and Sons, 2011.

[6] J. M. Nordbotten. *Finite volume methods*. In B. Engquist (Ed.) Encyclopedia of Applied and Computational Mathematics. Springer, 2016

[7] J. M. Nordbotten and E. Keilegavlen. *Inexact linear solvers for multi-level control volume discretization.* (In review) SIAM J. Sci. Comput.

[8] Y. Saad and M.H. Shultz. *GMRES: A General Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems.* SIAM J. Sci. Stat. Comput., 7(3): 856-869, 1986

[9] P. Sonneveld and M.B. van Gijzen. *IDR(s): a Family of simple and fast algorithms for solving large nonsymmetric linear systems.* SIAM J. Sci Comput., 31(2): 1035-1062, 2008.

[10] P. Sonneveld. *On the convergence behavior of IDR(s) and related methods.* SIAM J. Sci. Comput., 34(5): A2576-A2598, 2012.

[11] P. Sonneveld. *IDR(S) implementation manual.* Available at: http://ta.twi.tudelft.nl/nw/users/gijzen/IDR.html

[12] L.N. Trefethen and D. Bau. *Numerical Linear algerbra.* Society for Industrial and Applied Mathematics, 1997.

[13] H. van der Vorst. *Iterative Krylov Methods for Large Linear Systems.* Cambridge University Press, 2003