Development of a deflation-based linear solver in reservoir simulation

Joost van der Linden

Master of Science Thesis

TU Delft Delft University of Technology

Delft Institute for Applied Mathematics

Development of a deflation-based linear solver in reservoir simulation

Master of Science Thesis

For the degree of Master of Science in Applied Mathematics at
Delft University of Technology

Joost van der Linden

November 25, 2013

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
Delft University of Technology

The work in this thesis was supported by Schlumberger Oilfield UK Plc. Their cooperation is hereby gratefully acknowledged.

Delft University of Technology
Department of
Delft Institute of Applied Mathematics (diam)

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) for acceptance a thesis entitled

Development of a deflation-based linear solver in reservoir simulation

by

Joost van der Linden

in partial fulfillment of the requirements for the degree of

Master of Science Applied Mathematics

Dated: November 25, 2013

Committee members:

Prof.dr.ir. C. Vuik

Dr.ir. T.B. Jönsthövel

Prof.dr.ir. H.X. Lin

Abstract

Extreme and isolated eigenvalues are known to be harmful to the convergence of the iterative solver. These eigenvalues are produced by strong heterogeneity in the underlying physics. We can improve the quality of the spectrum by 'deflating' the harmful eigenvalues. In this thesis, deflation is applied to linear systems in reservoir simulation. We show that large, sudden differences in the permeability produce extreme eigenvalues. For small cases, the number and magnitude of these eigenvalues is linked to the number and magnitude of the permeability jumps. Two deflation methods are investigated. Firstly, we show that harmonic Ritz eigenvector deflation, which computes the deflation vectors from the information produced by the linear solver, can improve convergence. The computational cost of this method, however, is relatively high and the method cannot be implemented in parallel. Secondly, we test a physics-based deflation algorithm that constructs the deflation vectors a priori. The method is shown to improve the performance of the linear solver. We compare manually constructed deflation vectors to a partitioner algorithm, which automatically identifies large permeability jumps and constructs the deflation vectors using the subdomain-levelset method. Automatic (parallel) physics-based deflation works well for small cases, but we also show that the partitioner algorithm is not robust in large, realistic cases. We make several suggestions for improvement, such as assigning deflation vectors only to regions where flow occurs. For cases with well-defined permeability jumps of a factor 10^4 or higher, we believe that physics-based deflation has a large potential.

Contents

List of Figures

List of Tables

Preface & acknowledgments

The practical research in this thesis was conducted in the United Kingdom, at the Schlumberger Abingdon Technology Center (AbTC). Schlumberger developers at AbTC provide support for existing reservoir simulation software, such as Eclipse. Furthermore, to meet the increased demand for fast, scalable software, Intersect has been developed as the successor of Eclipse. Under the supervision of Dr. Tom Jönsthövel, I spent six months working at AbTC on the Intersect engine. On behalf of the Delft University of Technology, the research was supervised by Prof. Kees Vuik.

During my time in the United Kingdom, I was given the opportunity to attend the 'International Conference On Preconditioning Techniques For Scientific And Industrial Applications,' held at Oxford University. I subsequently reported the latest developments to the global applied mathematics community in Schlumberger. Furthermore, I presented my progress on an internal applied mathematics seminar at AbTC. Lastly, I presented our work, through teleconference, on a high-performance computing workshop held in Houston, USA.

The topic of this thesis is deflation. Intuitively, deflation can be understood as a method to improve convergence of the iterative solver by removing harmful components from the discretized partial differential equation. In particular, the iterative solver encounters difficulties with reservoirs that exhibit strong heterogeneity. The large jumps of, in this case, the permeability produce the harmful components (extreme eigenvalues). With deflation, these components can be targeted and removed from the iterative solve. As will be demonstrated, the effectiveness of deflation hinges on the ability to identify the permeability jumps.

Conducting research on deflation has been a profoundly enriching experience. First and foremost, I would like to thank Tom for his efforts in making this research a success. After a warm welcome, Tom has continuously shown passion and enthusiasm for our collaboration. I am deeply indebted to him for his helpful suggestions and answers to my questions. Our discussions were both worthwhile and inspiring. In particular, I am grateful to Tom for his endless patience in introducing me to C++ and Intersect.

Furthermore, I am thankful that Tom encouraged me to participate and present in a conference at Oxford University, an internal seminar and an external workshop. These events have been very helpful for my future career. In addition to the research, I have learned a lot from Tom's views on doing a PhD and working in industry. His experiences have helped me set out a path for the future. A warm welcome was followed by a warm goodbye, for which I would also like to thank the future Mrs. Jönsthövel. I wish you both a very happy life together.

I am grateful for the supervision of Prof. Kees Vuik. His effort to steer the research in the right direction, and his consistent feedback on my updates, has been very helpful. It was a privilege to attend the preconditioning conference in Oxford together. I would like to thank Dr. Alex Lukyanov for his interest in my research. We identified an overlap in our research (meshless deflation methods), which resulted in an interesting collaboration. Alex's feedback on the theoretical sections of this report, as well as his invitation to speak at the internal seminar, is much appreciated. It has been a privilege to work with my colleagues at AbTC. I owe much gratitude to my manager, Gareth Shaw, for his input on the project, his guidance for the coding and for the lunch on my last day. I thank Schlumberger for the financial support during my time in the United Kingdom. The trip would not have been possible without the support of my parents either, to whom I also dedicate my academic achievements. Last but not least, I thank my future wife, Katie Sundberg. She has sacrificed a lot to join me on my journeys, for which I am eternally grateful. I cannot imagine a better partner in life.

Delft, University of Technology — Joost van der Linden
November 25, 2013

Chapter 1

Introduction

Recent developments in the petroleum industry lead to the challenge of managing more data, providing higher field resolutions and computing more accurate multiphase flow predictions. The geological formations in petroleum reservoirs are subject to complex geometries and high physical contrasts, which also imposes constraints on the development. At the same time, advancements in hardware, such as (cheap) parallel systems and GPU-acceleration, demand the development of new algorithms that exceed previous performance records. This continuous interplay between computational demand and supply has led to initial development of the Intersect (IX) software. Through a joint venture of Schlumberger, Chevron and Total, the next-generation reservoir simulation software is continuously adapted to account for more complex geology and wells. IX is based on the company's previous, yet still widely used, reservoir simulator Eclipse. Over the last decade, IX has been designed to include a multi-segment well model, a scalable parallel computing infrastructure and a complete field management workflow [1]. Results are visualized with Schlumberger's model environment Petrel.

At the core of any reservoir simulator is the solver mechanism. A non-linear solver progresses the solution over time, constructing the Jacobian and solving a set of linear problems at each time step. The linear solver is an iterative method, usually preconditioned to counter the severe ill-conditioning effects caused by the highly heterogeneous porous media flow. The preconditioner in IX decouples the linear system in two sets of equations, exploiting the specific properties of each decoupled system. In this report, we will investigate the potential of an additional preconditioning technique: deflation. By removing unfavorable eigenvalues from the spectrum of our linear system, the deflation preconditioner can be used to improve convergence.

Deflation was first proposed for symmetric linear systems and the CG method by Nicolaides [23] and Dostál [24]. Both construct a deflation subspace consisting of deflation

vectors to deflate unfavorable eigenvalues from the linear system. A range of widely used deflation algorithms have been developed since, differing primarily in the method of application of the deflation operator and the approach to construct the deflation vectors. Deflation has been successfully used in a large number of applications, including, for example, electromagnetics [25], bubbly flow [26, 27, 28, 29, 30, 31, 32], structural mechanics and composite materials [33, 34, 35, 36], unsteady turbulent airfoil problems [20] and wave models in ship simulations [37]. The work by Vuik and co-authors on layered problems in reservoir simulation [8, 11, 38, 39, 40, 41] is the most relevant for this thesis.

The two most prominent deflation methods used in this work are:

1. **Harmonic-Ritz eigenvector deflation** uses approximate eigenvectors as deflation vectors in a black-box type of method. The deflation vectors are obtained after a full cycle of the linear solver. As will become clear, this imposes a severe computational burden on the software. We investigate if the costs outweigh the benefit of deflating a number of extreme eigenvalues.

2. **Physics-based deflation** circumvents the approximation by constructing the deflation a priori. Central to the investigation by Vuik and co-authors is the relation between the occurrence of extreme eigenvalues and large jumps in the PDE coefficients. In [11], for example, the number of extreme eigenvalues is proven to be equal to the number of high-permeability layers (e.g. sand) between low-permeability layers (e.g. shale). Having observed this, the question arises how to utilize the predictable spectrum in layered problems. In [11] and subsequent work, it is shown that the subspace spanned by the eigenvectors corresponding to the extreme eigenvalues can be approximated by a pre-determined space of algebraic deflation vectors. Convergence of the deflated CG method is shown to be independent of the size of the jumps in the coefficients.

We extend the work by Vuik and co-authors to non-symmetric linear systems in IX. Moreover, we will show that the particular physics-based deflation method used in this study allows for an efficient parallel implementation. Numerical experiments in IX, using a number of cases with varying size and degree of complexity, are used to analyze the performance.

1-1 Scope

In this thesis, an extensive overview will be provided of the current linear solver in IX, discussing formulation of the linear system, the Newton-Raphson method, GMRES and the two-stage AMG/ILU preconditioner. Deflation will be introduced through a discussion of variations of the method, the mathematical derivation, and the techniques to select the deflation vectors. After narrowing the focus to the two most promising approaches, we discuss the implementation. The first method, harmonic Ritz deflation, is

implemented in serial. The second method, a physics-based subdomain-levelset deflation algorithm, is coded both in serial and in parallel. Both methods are first prototyped in Matlab for validation, followed by a C++ implementation in IX. Finally, we explore if deflation can be used to speed up the linear solver.

1-2 Outline

The chapter overview of this report is as follows:

Chapter 2: Framework. The Intersect engine consists of a non-linear solver and a preconditioned linear solver. In Chapter 2, we discuss each individual component: Newton-Raphson, Arnoldi, GMRES and CPR preconditioning.

Chapter 3: Deflation. In Chapter 3, we introduce deflation. We motivate the use of deflation using an example, in which we link natural deflation to the convergence of the Ritz values. Natural deflation gives rise to artificial deflation, which we examine in detail. Several methods to compute the deflation vectors are investigated in terms of advantages and disadvantages.

Chapter 4: Implementation. The implementation of the preferred deflation methods from Chapter 3, harmonic Ritz deflation and subdomain-levelset deflation, is presented in Chapter 4. We introduce a partitioner algorithm to construct the deflation vectors a priori, and discuss the parallel implementation of deflation.

Chapter 5: Results. The results of our numerical experiments are presented in Chapter 5. Using five cases with varying dimensions and complexity, we test and compare harmonic Ritz deflation and subdomain-levelset deflation. We use the number of pressure iterations and the CPU time for the linear solve to draw conclusions about the speed and robustness of the methods.

Chapter 6: Conclusion. In the last chapter, we summarize the most important results and present suggestions for future research.

1-3 Notation and conventions

Table 1-3.1 lists the elementary notation that will be used throughout the report:

Notation	Meaning
I	Identity matrix with appropriate dimension
e_k	k'th natural basis vector
$1eP$	1×10^P
$\bar{H}_m$	Matrix obtained from H_m by omitting the last row
$\hat{x}$	Solution of the deflated linear system
$\mathcal{P}_m$	Set of polynomials of degree at most m
$A_{[i,j]}$	The (i,j)'th sub (block) matrix of the matrix A
a_{ij}	The (i,j)'th element of the matrix A
v_i or $v[i]$	The i'th element of the vector v (context provided)

Table 1-3.1: Standard notation.

Vectors are always denoted without an overhead arrow or bold script. Context will be provided whenever this can cause confusion with a scalar.

The units for the most commonly-used variables in this report are given in Table 1-3.2.

Variable	Unit	Name
Permeability	mD ($10^{-15}m^2$	Milidarcy
Porosity	m^3/m^3	Void space / total volume
Pressure	Pa ($kg/(m \cdot s^2)$)	Pascal
Temperature	$^\circ C$	Degree Celsius

Table 1-3.2: Units.

Chapter 2

Framework

To outline the mathematical framework in this report, we use IX to guide the discussion. Figure 2-0.1 provides an overview of the IX engine.

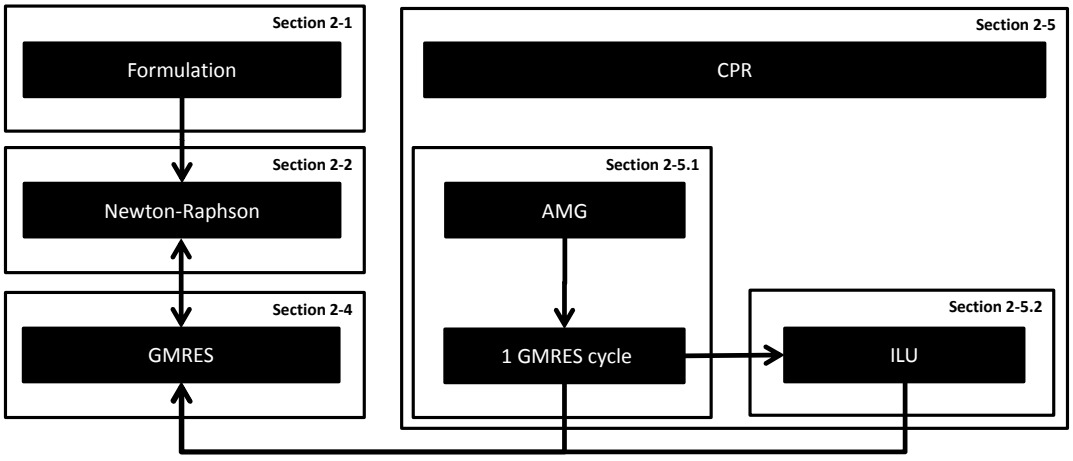


Figure 2-0.1: IX overview

In the simulation of a reservoir, the IX engine utilizes the linear and non-linear solver to progress the solution over time. For a given time step, GMRES solves the linear system produced by the Newton-Raphson method. GMRES is preconditioned by the constrained pressure residual (CPR), which consists of two stages to individually tackle the elliptic (stage one) and hyperbolic (stage two) part of the discrete operator [43]. The first stage of the CPR preconditioner restricts the linear system to the pressure

equations, followed by the construction of a GMRES-accelerated AMG preconditioner. Stage two entails an incomplete LU (ILU) factorization applied to the complete system. Since the stage two preconditioner uses the pressure solution, an arrow in Figure 2-0.1 connects GMRES in stage one to the ILU preconditioner in stage two.

In the upcoming subsections, we will discuss each component of the IX engine in detail.

2-1 Formulation

We focus on the reservoir equation formulation, and refer to the IX Technical Documentation [2] for more details on modeling of aquifer[1], thermal effects and wells. A general phase-component partitioning, called the "general formulation", has been used in IX to produce a single, unified code for the reservoir physics [3]. A number of components (e.g. oil, water, gas) can exist in a number of phases (e.g. solid, liquid, gas). Rather than having custom extensions for different phase- and component mixes, the general formulation allows for any number of phases and components. Each component can exist in any state and no particular ordering is assumed. For the simulator, the only input necessary is to specify which components can exist in which phase.

For illustrative purposes, we consider the case of two-phase (oil and water) flow. For a discussion of this model, see for example [42]. The assumptions are outlined next.

Assumption 2-1.1.

- *The flow is immiscible, i.e. the components do not dissolve.*
- *Fluids and rock are incompressible, i.e. the density remains constant.*
- *The effect of cappillary action, i.e. the ability of a liquid to flow against gravity in a narrow space such as pores, is ignored.*
- *Gravity effects in the rock and fluids are ignored.*

The governing conservation equations for the two-phase flow under these assumptions are:

$$\begin{cases} \phi \frac{\partial S_w}{\partial t} + \nabla \cdot (\lambda_w \nabla p) = -q_w, \\ \phi \frac{\partial S_o}{\partial t} + \nabla \cdot (\lambda_o \nabla p) = -q_o. \end{cases} \tag{2-1.1}$$

Subscripts w and o are used to indicate water and oil, respectively. The variable ϕ represents the porosity, or the ratio of void spaces (pores) over the total volume. The porosity is multiplied with the derivative of the saturation S. In the second term of the left-hand-side, λ is called the phase mobility, discussed below, and p is the pressure. The variable q in the right-hand-side gives the contribution of sources and/or sinks in the

[1] Aquifers are underground layers of water-bearing permeable rock or soil

reservoir. Summing the equations in (2-1.1) and using the fact that $S_w + S_o = 1$, we obtain the pressure equation

$$\nabla \cdot \lambda \nabla p = q, \tag{2-1.2}$$

where $\lambda = \lambda_w + \lambda_o$ is the total mobility and $q = q_w + q_o$. Because the mobility is the PDE coefficient possibly containing severe discontinuities, the discretized pressure equation will be central to our discussion on deflation algorithms. The total mobility is computed by summing

$$\lambda_w = k(x)\frac{k_{r_w}(S_w)}{\mu_w}, \text{ and} \tag{2-1.3}$$

$$\lambda_o = k(x)\frac{k_{r_o}(S_o)}{\mu_o}. \tag{2-1.4}$$

Here, μ is the viscosity[2], k is the absolute permeability[3] and k_r is the relative permeability. The absolute permeability is dependent on the geological structure of the reservoir, usually determined through some form of geophysical imaging. As the absolute permeability is determined a priori, we often refer to k as the initial permeability. The relative permeability, on the other hand, depends on the saturation and will vary throughout the simulation. Both k and k_r may exhibit large jumps, although the initial permeability will often be dominant [42].

The overall flow problem is handled by solving (2-1.2) for p, and using Darcy's law[4] to obtain the fluid velocity $u = -\lambda \nabla p$. The saturations can then be computed from (2-1.1). Since the mobility depends on the saturation, the combined nonlinear system of equations is coupled.

The IX formulation is more complex than the two-phase framework presented here. Extra terms are included for additional components, such as natural gas, and omitting the assumptions in 2-1.1 requires a number of supplementary equations. When a component can exist in more than one phase, the phase equilibrium equation is needed. This relation between a pair of phases describes the balance in fugacity[5]. In addition, phase saturation and mole fraction constraints are used to eliminate variables and reduce the overall system size. Although used in implicit form, the pressure equation (2-1.2) in IX is very similar to (2-1.2), and the contribution of the mobility λ will also be present. This has important implications for our discussion on deflation methods, where we aim to exploit knowledge about k, k_r and μ to construct the deflation vectors.

[2]The viscosity is a measure of resistance to deformation, or the 'thickness' of the fluid.

[3]The permeability, usually determined by lab experiments, is proportional to rate of flow in a porous media.

[4]Darcy's equation describes the ability of a component to flow through porous media. The latter depends on the difference in pressure and permeability of the media between two points, as well as the distance.

[5]Fugacity can be described as the 'escaping tendency' of a component from one phase to another.

The discretization, which is done in IX using the finite volume method, will not be further discussed in this report. After inclusion of the well equations and several steps of reduction, the coupled linear system can be expressed as

$$Ax = \begin{bmatrix} A_{rr} & A_{rw} \\ A_{wr} & A_{ww} \end{bmatrix} \begin{bmatrix} x_r \\ x_w \end{bmatrix} = \begin{bmatrix} b_r \\ b_w \end{bmatrix} = b. \tag{2-1.5}$$

Remark 2-1.1.

- *The subscript r refers to reservoir and w stands for well. We assume $A \in \mathbb{R}^{n \times n}$, $b_r, x_r \in \mathbb{R}^{nr}$, $b_w, x_w \in \mathbb{R}^{nw}$. The sub-matrices A_{rr} and A_{ww} are square $nr \times nr$ and $nw \times nw$ matrices respectively.*
- *Each sub-matrix of the Jacobian represents a derivative, e.g. A_{wr} is the derivative matrix of the well equations with respect to the reservoir variables.*

As not every cell will contain all components in all phases at all times, the order and number of equations and variables in the linear system may vary per time step. Regardless, A is typically very sparse. For example, in a three-dimensional $6 \times 3 \times 3$ grid, the matrix Arr is illustrated in Figure 2-1.1. Squares indicate the block structure of A.

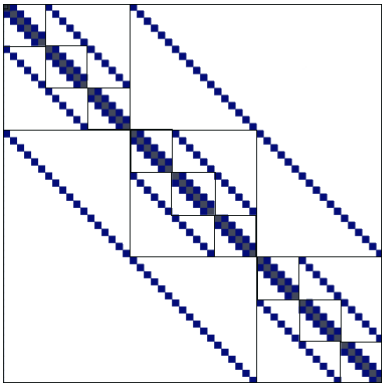


Figure 2-1.1: A_{rr} on a $6 \times 3 \times 3$ grid [2].

As shown in the two-phase flow, the coupled system of equations consists of two parts: the pressure equation and the saturation or mass balance equations. The former tends to be elliptic, because of short-range coupling, while the latter typically have a more global dependency and are therefore hyperbolic.

The linear solver, discussed in Section 2-3, is not directly applied to (2-1.5). Instead, the Schur complement is used:

$$(A_{rr} - A_{rw}A_{ww}^{-1}A_{wr})x_r = b_r - A_{rw}A_{ww}^{-1}b_w. \tag{2-1.6}$$

x_w can then be recovered by computing

$$x_w = A_{ww}^{-1}(b_w - A_{wr}x_r).$$

In the upcoming sections, we will refer to $Ax = b$ using

$$\begin{aligned} A &:= A_{rr} - A_{rw}A_{ww}^{-1}A_{wr}, \\ b &:= b_r - A_{rw}A_{ww}^{-1}b_w. \end{aligned} \tag{2-1.7}$$

Throughout this report, we assume that matrix A has the following properties:

- The computational grid consists of n_c cells or 'points', and each cell contains n_u unknowns. Hence, $n = n_c \times n_u$.
- $A \in \mathbb{R}^{n \times n}$ as given by (2-1.6).
- $A \neq A^T$, i.e. A is non-symmetric.
- $\lambda \neq 0 \; \forall \; \lambda \in \sigma(A)$, i.e. all eigenvalues are non-zero.

2-2 Newton-Raphson

Suppose $F(x)$ is a differentiable vector function from some interval in $\mathbb{R}^n$ to $\mathbb{R}^n$. Newton-Raphson attempts to solve the general non-linear system

$$F(x) = 0, \qquad x \in \mathbb{R}^n. \tag{2-2.1}$$

F represents the equations of the coupled system of reservoirs, wells and aquifers. The input vector x contains the solution variables for the pressure and the saturation. A Taylor expansion of (2-2.1) around the point x yields

$$F(x + \Delta x) = F(x) + \Delta x \cdot \nabla F \mid_x + \frac{1}{2}(\Delta x \cdot \nabla^2)F \mid_x + \mathcal{O}(\Delta x^3),$$

where Δx is the difference between x and an arbitrary point close to x. Setting this expression equal to zero, ignoring higher-order terms and rearranging gives us

$$\nabla F(x)\Delta x = -F(x).$$

An iterative method is obtained by letting $\Delta x = x_{j+1} - x_j$, i.e.

$$\nabla F(x_j)\Delta x = -F(x_j), \tag{2-2.2}$$

where, after this system is solved for Δx, $x_{j+1} = x_j + \Delta x$. Note that after linearization, Equation (2-2.2) corresponds to the the coupled linear system (2-1.5). In each iteration of Newton-Raphson, the Jacobian matrix is formed and (2-1.5) is solved for x using the linear solver described in the next section.

2-2-1 Time step size

The convergence criterion for Newton-Raphson in IX is based on a dynamic mix of the absolute solution error, overall mass balance error and the minimum and maximum number of iterations. After convergence, a new time step is selected. The length of this step is determined by a number of selection criteria:

- The minimum and maximum time step, and the ratio of increase compared to the previous time step.
- The first upcoming 'report date', i.e. the user-requested time at which the solution should be computed. Instead of taking a large step followed by a very short step to reach the report date, it can be more beneficial to take two steps of equal length.
- The time truncation error limit, which prevents propagation of the truncation error due to linearization.
- The maximum rate of change for the solution. Each of the solution variables (pressure, saturation, thermal) is bounded by a maximum change criteria, to guarantee convergence.
- The Courant-Friedrichs-Lewy (CFL) stability criterion. Unless a fully implicit simulation is used, the time step size is restricted by the CFL condition, to guarantee convergence of the full solution.

In practice, a balance is required between implicit and explicit formulation of the different variables (e.g. pressure, saturation). For increased implicitness, we obtain better stability. Larger timesteps can be taken, but the cost of the linearization and the linear solve increase quadratically. In practice, the Adaptive Implicit Method (AIM) is used. Using the CFL condition, each cell is tested for stability. The most unstable cells are treated fully implicit, whereas IMPES (implicit pressure, explicit saturations mole fractions) and IMPSAT (implicit pressure and saturations, explicit mole fractions) are used for the variables in more stable cells. The formulation of the variables is beyond the scope of this thesis. Therefore, a fully implicit model is used for all cells, which is unconditionally stable.

2-3 Krylov-subspace methods

Consider the coupled linear system

$$Ax = b \tag{2-3.1}$$

satisfying the properties outlined in section 2-1. A common denominator for most existing iterative techniques is the use of a certain projection method, arising from the Petrov-Galerkin conditions [4]. Two m-dimensional subspaces of $\mathbb{R}^n$ are used in this context: $\mathcal{K}_m$, which is referred to in literature as the search, solution or ansatz space,

and $\mathcal{L}_m$, which is called the test or constraints space. The subspace $\mathcal{K}_m$ contains the candidate approximations for the solution of (2-3.1), while $\mathcal{L}_m$ accommodates a set of m orthogonality constraints. The Petrov-Galerkin conditions state that the residual $r = b - Ax$ should be orthogonal to $\mathcal{L}_m$.

Let x_0 be an initial guess for the approximate solution x_m of (2-3.1), and denote the exact solution by x^*. Given $\mathcal{K}_m$ and $\mathcal{L}_m$, the Petrov-Galerkin conditions imply

$$\text{Find } x_m \in x_0 + \mathcal{K}_m \text{ such that } b - Ax_m \perp \mathcal{L}_m. \tag{2-3.2}$$

This expression can be translated in a general procedure for Krylov subspace methods.

1. Construct an $n \times m$ orthonormal basis V_m of $\mathcal{K}_m$.
2. Construct an $n \times m$ basis W_m of $\mathcal{L}_m$.
3. Approximate the solution of (2-3.1) by $x_m = x_0 + V_m u_m$, where $u_m \in \mathbb{R}^m$ is a weights vector, while imposing the Petrov-Galerkin conditions $W_m^T(b - Ax_m) = 0$.

Substitution of $x_m = x_0 + V_m u_m$ in step 3 and rearranging yields

$$\begin{cases} x_m = x_0 + V_m u_m, \\ u_m = (W_m^T A V_m)^{-1} W_m^T r_0. \end{cases} \tag{2-3.3}$$

In the next section an efficient algorithm to compute a basis for $\mathcal{K}_m$ will be presented. The Generalized Minimal Residual (GMRES) method, proposed by Saad and Schultz [46], is essentially a clever way of solving an m-dimensional linear system equivalent to (2-3.3) [6].

For the particular case of GMRES, the orthogonality constraints are given by $\mathcal{L}_m = A\mathcal{K}_m$. A closely related approach for symmetric (semi-)positive definite matrices is the Conjugate Gradient (CG) method, in which case each search direction is also an orthogonality constraint: $\mathcal{L}_m = \mathcal{K}_m$. In fact, the GMRES method and the CG method are part of broader classes of projection methods, known as (Galerkin) orthogonal residual methods and minimal residual methods, respectively [47]. For nonsingular Hermitian indefinite matrices, for example, the MINRES method is also a minimal residual method. Both the GMRES and the CG method are called Krylov-subspace methods, for their solution subspaces $\mathcal{K}_m$ are of the particular form presented in Definition 2-3.1.

Definition 2-3.1. *Let $A \in \mathbb{R}^{n\times n}$ and $r_0 \in \mathbb{R}^n$, given by (2-3.1) and $r_0 = b - Ax_0$ respectively. Then, for $m \leq n$, a Krylov subspace is defined as a subspace $\mathcal{K}_m$ of the form*

$$\mathcal{K}_m(A, r_0) = \text{span}\{r_0, Ar_0, A^2 r_0, \ldots, A^{m-1} r_0\}. \tag{2-3.4}$$

A Krylov subspace can also be understood by first applying the splitting $A = M - N$, followed by a rearrangement of (2-3.1) which yields the basic iterative method

$$Mx_{m+1} = b + Nx_m.$$

Using $r_m = b - Ax_m$, the iterative method becomes $x_{m+1} = x_m + M^{-1}r_m$. Developing the iterations, we find

$$\begin{cases} x_0 & \\ x_1 = & x_0 + M^{-1}r_0 \\ x_2 = & x_0 + 2M^{-1}r_0 - M^{-1}AM^{-1}r_0 \\ x_3 = & x_0 + 3M^{-1}r_0 - 3M^{-1}AM^{-1}r_0 + (M^{-1}A)^2M^{-1}r_0 \\ \vdots & \\ x_{m+1} = & x_0 + p_m(A)r_0 \end{cases} \tag{2-3.5}$$

where $p_{m+1} \in \mathcal{P}_{m+1}$ is a polynomial of degree $m + 1$. Hence,

$$x_{m+1} \in x_0 + \mathcal{K}_m(A, r_0)$$

Stationary Krylov methods apply the same operations every iteration. In the Jacobi method, for instance, M is fixed as the main diagonal of A. Other examples of stationary Krylov methods are Gauss-Seidel and (Symmetric) Successive Overrelaxation. The non-stationary counterpart, on the other hand, has iteration-dependent coefficients, as will become clear when we discuss the GMRES method.

Remark 2-3.1. *If we assume $x_0 = 0$, note that $A^{-1}b \approx x_{m+1} = p_m(A)b$. In other words, we obtain an approximation for the inverse of the matrix A. This observation relates back to the Cayley-Hamilton theorem, which implies that the inverse of a matrix can be expressed in terms of a linear combination of its powers.*

In the upcoming sections, we will denote Krylov subspaces as

$$\mathcal{K}_m(A, v) = \text{span}\{v, Av, A^2v, \ldots, A^{m-1}v\}.$$

Before we can establish the fundamentals of GMRES, it is necessary to first discuss one of its main components.

2-3-1 Arnoldi

To introduce the Arnoldi algorithm, we first discuss the power method [44]. This technique, also known as the Von Mises iteration, is used as an eigenvalue algorithm. When a vector is repeatedly multiplied by the same matrix, the eigenvector corresponding to

the largest eigenvalue of the matrix will dominate the result. Hence, given some starting vector v_1, the power method uses the iteration

$$v_{j+1} = \frac{Av_j}{\|Av_j\|_2}.$$

The sequence v_1, Av_1, A^2v_1, ..., iteratively normalized and stored, converges to the eigenvector corresponding to the largest eigenvalue of A. Convergence of the power method can be slow, and only the dominant eigenvector of A is computed [5]. The Arnoldi method is a variation of the power method that does not has these negative properties. Starting with the vector v_1 such that $\|v_1\|_2 = 1$, we orthonormalize Av_j, $j = 1, \ldots, m$ against all previous entries using an orthogonalization scheme.

The classical Gram-Schmidt orthogonalization scheme is often replaced by the Modified Gram-Schmidt (MGS) variant. In the latter case vectors are orthogonalized against errors in the computation of all previous orthogonalization steps. As a result, MGS produces the same result as GS in exact arithmetic, but with smaller errors in finite-precision computations. The pseudo-code of the Arnoldi method using MGS is given in Algorithm 1.

Algorithm 1 Arnoldi

1: Select v_1 such that $\|v_1\|_2 = 1$.
2: **for** $j = 1, 2, \ldots, m$ **do**
3: $w_j = Av_j$
4: **for** $i = 1, \ldots, j$ **do**
5: $h_{i,j} = (w_j, v_i)$
6: $w_j = w_j - h_{ij}v_i$
7: **end for**
8: $h_{j+1,j} = \|w_j\|_2$
9: **if** $h_{j+1,j} = 0$ **then** break
10: $v_{j+1} = w_j / h_{j+1,j}$
11: **end for**

Remark 2-3.2.

- *The m matrix-vector products Av_j are generally the primary cost of the algorithm.*
- *In the implementation w_j does not have to be stored, and it is often replaced by w. Consequently, the variable is overwritten in every Arnoldi iteration. Storage requirements include $n(m+1)$ elements for the Arnoldi vectors v_j, as well as $m(m+1)$ elements for the matrix $H = h_{ij}, 1 \leq i \leq m+1, 1 \leq j \leq m$.*
- *Line 9 of Algorithm 1 is included to prevent a division by zero. If $h_{j+1,j} = 0$, though, we will see shortly that the solution at iteration j will be the exact solution of* (2-3.1).

A useful property of the Arnoldi algorithm is proved next.

Proposition 2-3.1 (cf. [4, Prop. 6.4]). *Assume $h_{j+1,j} \neq 0$ for $j = 1, 2, \ldots, m$. Then the vectors $v_1, v_2, \ldots, v_m$ in Algorithm 1 constitute an orthonormal basis of of the Krylov subspace*

$$\mathcal{K}_m(A, v_1) = \text{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\}.$$

Proof. Since Algorithm 1 uses the MGS orthonormalization scheme, the vectors $v_j, j = 1, 2, \ldots, m$ are orthonormal by construction. To show that they are a basis of $\mathcal{K}_m(A, v_1)$, we need to prove that $v_j = p_{j-1}(A)v_1$ for all j. A simple induction argument can be used in this case. For $j = 1$, clearly, $v_1 = p_0(A)v_1$, where the polynomial p_0 is equal to one. Assuming the induction hypothesis $v_j = p_{j-1}(A)v_1$, marked $(*)$, we get

$$\begin{aligned} v_{j+1} = \frac{w_j}{h_{j+1,j}} &= \frac{1}{h_{j+1,j}} \left[w_j - \sum_{i=1}^{m} h_{ij} v_i \right] \\ &= \frac{1}{h_{j+1,j}} \left[Av_j - \sum_{i=1}^{m} h_{ij} v_i \right] \\ &\overset{(*)}{=} \frac{1}{h_{j+1,j}} \left[p_j(A)v_1 - \sum_{i=1}^{m} h_{ij} p_{i-1}(A)v_1 \right], \end{aligned}$$

so we find that v_{j+1} is of the form $p_j(A)v_1$, as required. □

Proposition 2-3.1 will be employed in the next section to find an approximate solution of the coupled linear system (2-3.1).

The Arnoldi algorithm produces the following matrices:

$$V_{m+1} = [v_1, v_2, \ldots, v_m] \text{ and } \bar{H}_m = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,m-1} & h_{1,m} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,m-1} & h_{2,m} \\ & h_{3,2} & \cdots & h_{3,m-1} & h_{3,m} \\ & & \ddots & \vdots & \vdots \\ & & & h_{m,m-1} & h_{m,m} \\ & & & & h_{m+1,m} \end{bmatrix} \tag{2-3.6}$$

The entries of the upper-Hessenberg matrix $\bar{H}_m$ are defined as $h_{i,j} = (Av_j)^T v_i$. V_{m+1} contains the basis vectors for the Krylov subspace $\mathcal{K}(v_1, A)$, as shown in 2-3.1. The following definition introduces Ritz values and Ritz vectors, which will play a crucial role in approximating eigenvectors.

Definition 2-3.2. *Let H_m be given as in Lemma 2-3.1. Then, the eigenvalues of H_m are called Ritz values. The eigenvector corresponding to a Ritz value, multiplied by V_m, is called a Ritz vector.*

In the next lemma, we derive two expressions concerning the matrices V_{m+1} and $\bar{H}_m$.

Lemma 2-3.1 (cf. [4, Prop. 6.5]). *Let V_{m+1} and $\bar{H}_m$ as in* (2-3.6). *Define $H_m \in \mathbb{R}^{m\times m}$ as the matrix obtained from $\bar{H}_m$ by omitting the last row. Then the following expressions hold:*

(a) $AV_m = V_{m+1}\bar{H}_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T$;

(b) $H_m = V_m^T A V_m$.

Proof. Part (a) follows by assembling

$$h_{m+1,m} v_{m+1} = w_m = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$$

for all $j = 1, 2, \ldots, m$, so that in matrix notation

$$h_{m+1,m} v_{m+1} e_m^T = AV_m - V_m H_m$$

from which the first identity of the proposition is readily derived. Multiplying part (a) by V_m^T on both sides proves part (b):

$$V_m^T A V_m = V_m^T V_m H_m + h_{m+1,m} V_m^T v_{m+1} e_m^T = H_m,$$

since $V_m^T V_m = I$ and $V_m^T v_{m+1} = 0$ by the orthogonality construction. □

Part (b) implies that Arnoldi maps the matrix A to an upper Hessenberg matrix H_m, using the orthogonality transformations given by V_m. From part (a) of Proposition 2-3.1 we find that in case of a breakdown, i.e. $h_{m+1,m} = 0$,

$$AV_m = V_m H_m.$$

As a result, the vectors in V_m form an invariant subspace of A. Proposition 2-3.2 shows the consequence of a lucky breakdown.

Proposition 2-3.2 (cf. [4, Prop. 5.6]). *Assume $\mathcal{K}_m$ is invariant under A and $r_0 = b - Ax_0 \in \mathcal{K}_m$. Then, the projection x_m will be the exact solution of $Ax = b$.*

Proof. Let W_m be the oblique projector onto $\mathcal{K}_m$, orthogonal to $\mathcal{L}_m$, as given in equation (2-3.3). The approximate solution x_m satisfies the Petrov-Galerkin orthogonality conditions

$$W_m^T (b - Ax_m) = 0. \tag{2-3.7}$$

Recall $x_m \in x_0 + \mathcal{K}_m$. Let $\delta \in \mathcal{K}_m$, then (2-3.7) yields

$$W_m^T (b - A(x_0 + \delta)) = 0 \;\Leftrightarrow\; W_m^T (r_0 - A\delta) = 0$$

By assumption, $W_m^T r_0 = r_0$. In addition, because $\delta \in \mathcal{K}_m$ and $\mathcal{K}_m$ is invariant under A, $A\delta \in \mathcal{K}_m$. Hence, $W_m^T A\delta = A\delta$. Equation (2-3.7) thus simplifies to

$$\begin{aligned} r_0 - A\delta = 0 &\;\Leftrightarrow\; b - Ax_0 - A(x_m - x_0) = 0 \\ &\;\Leftrightarrow\; b - Ax_m = 0, \end{aligned}$$

which shows that x_m is the exact solution. □

2-4 GMRES

Recall the Petrov-Galerkin conditions (2-3.2)

$$\text{Find } x_m \in x_0 + \mathcal{K}_m \text{ such that } (b - Ax_m, w) = 0 \ \forall \ w \in \mathcal{L}_m.$$

As noted before, the GMRES uses $\mathcal{L}_m = A\mathcal{K}_m$ for the constraints space. Hence, if we set $r_m = b - Ax_m$ as before, then the Petrov-Galerkin conditions imply $r_m \perp A\mathcal{K}_m$. In the upcoming theorem we show that this is equivalent to minimizing the residual.

Theorem 2-4.1 (cf. [6, Lem. 10.1])**.** *The Petrov-Galerkin conditions $r_m \perp A\mathcal{K}_m$ are equivalent to*

$$\|r_m\|_2 = \min_{x \in x_0 + \mathcal{K}_m} \|b - Ax\|_2 = \min_{r \in r_0 + A\mathcal{K}_m} \|r\|_2 \tag{2-4.1}$$

Proof. Let $x \in x_0 + \mathcal{K}_m$ arbitrary, with residual $r \in r_0 + A\mathcal{K}_m$, and denote $A\mathcal{K}_m = \mathcal{L}_m$. Write $r = r_0 + l$ with $l \in \mathcal{L}_m$. Define P_m as the orthogonal projector onto $\mathcal{L}_m$ and $Q_m = I - P_m$ as the projector on $\mathcal{L}_m^\perp$. We can then decompose r_0 as

$$r_0 = P_m r_0 + Q_m r_0 \in \mathcal{L}_m \oplus \mathcal{L}_m^\perp.$$

Hence, $r = (P_m r_0 + l) + Q_m r_0 \in \mathcal{L}_m \oplus \mathcal{L}_m^\perp$. Taking the two-norm and applying Pythagoras yields

$$\|r\|_2^2 = \|P_m r_0 + l\|_2^2 + \|Q_m r_0\|_2^2, \tag{2-4.2}$$

which attains its minimum when $l = -P_m r_0$. For this particular choice, the residual r_m is

$$r_m = r_0 - P_m r_0 = Q_m r_0 \in \mathcal{L}_m^\perp \perp \mathcal{L}_m,$$

i.e. $r_m \perp \mathcal{L}_m$ which was to be proven.

Assuming $r \perp \mathcal{L}_m$, on the other hand, implies $Q_m r = r$, as Q_m projects on $\mathcal{L}_m^\perp$, which already contains r. Therefore,

$$\|r\|_2^2 = \|Q_m r\|^2 = \|Q_m(r_0 + l)\|_2^2 = \|Q_m r_0\|_2^2,$$

since $Q_m l = 0$. Note that this is exactly the minimum value found from (2-4.2). □

The Arnoldi procedure to construct an orthonormal basis for the Krylov subspace $\mathcal{K}_m$ is the core of the Generalized Minimum Residual Method. The matrices H_m and V_m can be employed to compute the equivalent of (2-3.3) in an efficient way. In the next proposition, assume the Arnoldi method is applied with $v_1 = r_0/\|r_0\|_2$, to obtain a basis for the Krylov subspace $\mathcal{K}_m(A, v_1)$.

Proposition 2-4.1. *Let $\beta = \|r_0\|_2$ and u_m the weight vector in $x_m = x_0 + V_m u_m \in x_0 + \mathcal{K}_m$ by Proposition 2-3.1. Then the optimality criterion from Theorem 2-4.1 can be written as*

$$\min_{x \in x_0 + \mathcal{K}_m} \|b - Ax\|_2 = \min_{u \in \mathbb{R}^m} \|\beta e_1 - \bar{H}_m u\|_2. \tag{2-4.3}$$

Proof. First, note that $r_0 = \beta v_1 = \beta V_{m+1} e_1$. We derive,

$$\begin{aligned} b - Ax &= b - A(x_0 + V_m u) \\ &= r_0 - AV_m u \\ &= \beta V_{m+1} e_1 - AV_m u \\ &= \beta V_{m+1} e_1 - V_{m+1} \bar{H}_m u = V_{m+1}(\beta e_1 - \bar{H}_m u), \end{aligned}$$

where we used Lemma 2-3.1 in the last line. As a result,

$$\begin{aligned} \|b - Ax\|_2 &= \|V_{m+1}(\beta e_1 - \bar{H}_m u)\|_2 \\ &= \|\beta e_1 - \bar{H}_m u\|_2, \end{aligned}$$

because the columns of V_{m+1} are orthonormal. □

The obtained least-squares problem has a unique solution as long as $h_{j+1,j} \neq 0$, $j = 1, \ldots, m-1$, in which case $\bar{H}_m$ has full rank. For a minimum $u_m \in \mathbb{R}^m$ of (2-4.3), the approximation x_m to 2-3.1 is given by $x_m = x_0 + V_m u_m$.

Proposition 2-4.1 gives rise to the GMRES algorithm, stated next.

Algorithm 2 GMRES

1: Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, and $v_1 = r_0/\beta$.
2: **for** $j = 1, 2, \ldots, m$ **do**
3: $w_j = Av_j$
4: **for** $i = 1, \ldots, j$ **do**
5: $h_{i,j} = (w_j, v_i)$
6: $w_j = w_j - h_{ij} v_i$
7: **end for**
8: $h_{j+1,j} = \|w_j\|_2$
9: **if** $h_{j+1,j} = 0$ **or** converged **then**
10: set $m = j$ and **go to** 14
11: **end if**
12: $v_{j+1} = w_j / h_{j+1,j}$
13: **end for**
14: Fill $\bar{H}_m = \{h_{ij}\}$ for $1 \leq i \leq m+1$, $1 \leq j \leq m$.
15: Compute the minimizer u_m of $\|\beta e_1 - \bar{H}_m u\|_2$ and set $x_m = x_0 + V_m u_m$.

Remark 2-4.1.

- *Observe how the Arnoldi algorithm 1 is employed in two ways: to compute the basis vectors v_j for the Krylov subspace $\mathcal{K}_m(A, v_1)$, and to fill the upper Hessenberg matrix $\bar{H}_m$. The latter two corresponding matrices constitute the main memory requirements of the GMRES method.*
- *If the algorithm has not converged after m iterations, a restart can be performed by setting $x_0 = x_m$ and returning to line 1. This method is usually referred to as restarted GMRES, or simply GMRES(m) where m is the maximum number of iterations before a restart.*
- *In case restarted GMRES is used, the restart loop is called the 'outer' loop, and the computations for $j = 1, 2, \ldots, m$ are referred to as being in the 'inner' loop. A full run of the inner loop is called a 'cycle'. The variable m is often referred to as the 'cycle size'.*

Two components of the GMRES algorithm have not been explained yet. In the upcoming two subsections, we will present an effective method to compute the minimizer u_m, and show how to check for convergence.

2-4-1 Givens rotations

As shown in (2-3.6), the matrix $\bar{H}_m$ is upper Hessenberg. Because this particular matrix-shape is nearly upper diagonal, solving 2-4.3 by a QR-factorization is favorable. This gives us

$$\bar{H}_m = Q_m \bar{R}_m, \tag{2-4.4}$$

where Q_m is an $m+1 \times m+1$ orthogonal matrix, and

$$\bar{R}_m = \begin{bmatrix} R_m \\ 0 \end{bmatrix},$$

where $\bar{R}_m$ is $m+1 \times m$ and R_m is $m \times m$ upper triangular and invertible. In addition, let

$$\bar{g}_m = Q_m \beta e_1 = \begin{bmatrix} g_m \\ \gamma_{m+1} \end{bmatrix}$$

where γ_{m+1} is a scalar and $\bar{g}_m$ and g_m are vectors of length $m+1$ and m respectively. Note that because Q_m is orthogonal, (2-4.4) is equivalent to $Q_m \bar{H}_m = \bar{R}_m$. As a result, the norm in the right-hand-side of (2-4.3) becomes

$$\begin{aligned} \|\beta e_1 - \bar{H}_m u\|_2^2 &= \|Q_m(\beta e_1 - \bar{H}_m u)\|_2^2 \\ &= \|\bar{g}_m - \bar{R}_m u\|_2^2 \\ &= |\gamma_{m+1}|^2 + \|g_m - R_m u\|_2^2. \end{aligned} \tag{2-4.5}$$

The minimum of this expression is achieved when $\|g_m - R_m u\|_2^2 = 0$. Hence, the least-squares problem (2-4.3) can be tackled by solving $R_m u_m = g_m$ using back substitution.

The question that remains is: how do we obtain the QR-decomposition of $\bar{H}_m$? Recall that the matrix is upper Hessenberg, i.e. all entries below the first subdiagonal are zero. The QR-decomposition is thus obtained if we find an orthogonal matrix Q_m that eliminates the subdiagonal elements in $\bar{H}_m$, resulting in an upper diagonal matrix $\bar{R}_m$. This can be achieved using Givens rotations.

The θ radians counterclockwise rotation of a vector $y \in \mathbb{R}^m$ is defined as

$$G_{ij}^m(\theta) = \begin{array}{c} \\ 1 \\ \vdots \\ i \\ \vdots \\ j \\ \vdots \\ m \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & \cdots & i & \cdots & j & \cdots & m \end{array} \\ \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & \cdots & s & & \\ & & \vdots & \ddots & \vdots & & \\ & & -s & \cdots & c & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix} \end{array}$$

where $c = \cos(\theta)$, $s = \sin(\theta)$ and $c^2 + s^2 = 1$. The action of G on y is

$$G_{ij}^m(\theta) y = \tilde{y}; \qquad \tilde{y}_k = \begin{cases} cy_i + sy_j & \text{for } k = i \\ cy_j - sy_i & \text{for } k = j \\ y_k & \text{for } k \neq i, j \end{cases}.$$

Now if we choose

$$c = \frac{y_i}{\sqrt{y_i^2 + y_j^2}}$$
$$s = \frac{y_j}{\sqrt{y_i^2 + y_j^2}}$$

then $y_j = 0$, while preserving the norm of y. We will assume $j = i + 1$ and write G_i^m for the Givens rotation that eliminates $y_{i+1,i}$ from an arbitrary matrix $Y = \{y_{ij}\}$ with m columns. As a result, a series of Givens rotations $Q_m = G_m^m G_{m-1}^m \dots G_1^m$ will transform the upper Hessenberg matrix $\bar{H}_m$ into the upper triangular matrix $\bar{R}_m$:

$$Q_m \bar{H}_m = G_m^m G_{m-1}^m \dots G_1^m \begin{bmatrix} \bullet & \bullet & \cdots & \bullet \\ \bullet & \bullet & \cdots & \bullet \\ & \bullet & \cdots & \bullet \\ & & \ddots & \vdots \\ & & & \bullet \end{bmatrix} = \begin{bmatrix} \circ & \circ & \cdots & \circ \\ & \circ & \cdots & \circ \\ & & \ddots & \vdots \\ & & & \circ \\ & & & \end{bmatrix} = \bar{R}_m.$$

We obtain the matrices Q_m and $\bar{R}_m$ as described in (2-4.4).

Each inner iteration of GMRES adds one column to the upper Hessenberg matrix. For any $\bar{H}_j$, $1 \leq j \leq m-1$, the action of the first j Givens rotations on $\bar{H}_{j+1}$ is

$$G_j^{j+1} G_{j-1}^{j+1} \dots G_1^{j+1} \bar{H}_{j+1} = G_j^{j+1} G_{j-1}^{j+1} \dots G_1^{j+1} \begin{bmatrix} \bullet & \bullet & \cdots & \bullet & h_{1,j+1} \\ \bullet & \bullet & \cdots & \bullet & h_{2,j+1} \\ & \bullet & \cdots & \bullet & h_{3,j+1} \\ & & \ddots & \vdots & \vdots \\ & & & \bullet & h_{j+1,j+1} \\ & & & & h_{j+2,j+1} \end{bmatrix}$$
$$= \begin{bmatrix} \circ & \circ & \cdots & \circ & \tilde{h}_{1,j+1} \\ & \circ & \cdots & \circ & \tilde{h}_{2,j+1} \\ & & \ddots & \vdots & \vdots \\ & & & \circ & \tilde{h}_{j,j+1} \\ & & & & \tilde{h}_{j+1,j+1} \\ & & & & h_{j+2,j+1} \end{bmatrix}.$$

Note that the upper left $j \times j$ submatrix in the right-hand side is again $\bar{R}_j$, while in the rightmost column only $h_{j+2,j+1}$ remains unchanged. To obtain $\bar{R}_{j+1}$, we eliminate $h_{j+2,j+1}$ by using the Givens rotation G_{j+1}^{j+1} with

$$\begin{aligned} c &= \frac{h_{j+1,j+1}}{\sqrt{h_{j+1,j+1}^2 + h_{j+2,j+1}^2}} \\ s &= \frac{h_{j+2,j+1}}{\sqrt{h_{j+1,j+1}^2 + h_{j+2,j+1}^2}} \end{aligned} \tag{2-4.6}$$

The fact that $h_{j+2,j+1}$ is not affected by the first j Givens rotations, combined with the observation that G_{j+1}^{j+1} only affects the $j+1$'th column in $\bar{H}_{j+1}$, gives rise to the following iterative implementation of Givens rotations in GMRES:

1. Apply the previous rotations $G_j^{j+1} G_{j-1}^{j+1} \dots G_1^{j+1}$ to the first $j+1$ elements of the new column in $\bar{H}_{j+1}$, i.e. $h_{1,j+1}, \dots, h_{j+1,j+1}$.
2. Compute the new rotation G_{j+1}^{j+1} using (2-4.6), and apply to $\bar{H}_{j+1}$ and βe_1.
3. Repeat until convergence, breakdown or the end of a cycle.

This update of $\bar{H}_{j+1}$ is relatively cheap, i.e. $\mathcal{O}(j)$. Storage of the Givens rotation matrices can usually be avoided by saving the individual elements c and s in two vectors. Although it is possible to use the procedure above after the inner loop has finished, it is rather more efficient to directly apply the Givens rotations during each inner iteration.

2-4-2 Computing the residual

In Algorithm 2, the solution is not explicitly available in each iteration of the inner loop. Hence, computing the residual cannot be done by the straight-forward computation

$$\|r_m\|_2 = \|b - Ax_m\|_2 = \|\beta e_1 - \bar{H}_m u_m\|_2.$$

Instead, we will see next that the Givens rotations can be exploited to find the residual.

Proposition 2-4.2 (cf. [4, Prop. 6.9c]). *The norm of the residual vector is given by*

$$\|r_m\|_2 = |\gamma_{m+1}|$$

Proof. Recall from the proof of 2-4.1 that $b - Ax = V_{m+1}(\beta e_1 - \bar{H}_m u)$. Using the orthogonal matrix Q_m, we find

$$\begin{aligned} b - Ax &= V_{m+1}(\beta e_1 - \bar{H}_m u) \\ &= V_{m+1} Q_m^T Q_m (\beta e_1 - \bar{H}_m u) \\ &= V_{m+1} Q_m^T (\bar{g}_m - \bar{R}_m u). \end{aligned}$$

When u is computed as the minimizer of (2-4.5), we are left with

$$b - Ax = V_{m+1} Q_m^T (\gamma_{m+1} e_{m+1}),$$

or, in the two-norm,

$$\|b - Ax\|_2 = \|V_{m+1} Q_m^T (\gamma_{m+1} e_{m+1})\|_2 = \|\gamma_{m+1} e_{m+1}\|_2 = |\gamma_{m+1}|,$$

because the product of two orthogonal matrices is again orthogonal. □

As a convergence criterion we compare, unless otherwise stated, the current residual divided by the norm of the right-hand side against some predefined tolerance.

2-4-3 Convergence

Can the residual at each iteration be bounded? Unfortunately, this is not trivial for non-symmetric systems. Whereas strong convergence bounds are known for normal matrices, the known bounds on the residual in GMRES are not well defined. Moreover, non-symmetric matrices are rarely normal. In theory, Greenbaum, Pták and Strakos [45] show that, for any non-increasing convergence curve of the residual, there exists a matrix A and an initial residual such that GMRES will follow this particular curve. Moreover, the matrix A can be chosen to have any arbitrary eigenvalue distribution.

If A is normal, the relative residual can be bounded as

$$\frac{\|r_m\|}{\|r_0\|} \leq \min_{\substack{p\in\mathcal{P}_m \\ p(0)=1}} \max_k |p(\lambda_k)| \tag{2-4.7}$$

where $\mathcal{P}_m$ is the set of polynomials of degree at most m, satisfying $p(0) = 1$, and λ_k denotes the k'th eigenvalue. This bound is sharp, i.e. it describes the worst-case scenario. The relation between the bound in (2-4.7) and the eigenvalue distribution is nonlinear and rather complicated [47]. For different properties of A, simplifications can be used. If the largest and smallest eigenvalue are known in Hermitian positive definite matrices, for example, Chebyshev polynomials can be used to approximate the min-max problem for the CG method. The result is a bound on the relative A-norm of the error:

$$\frac{\|x^* - x_m\|}{\|x^* - x_0\|} \leq 2\left(\frac{\sqrt{\kappa_2(A)}-1}{\sqrt{\kappa_2(A)}-1}\right)^m, \qquad \kappa_2(A) = \frac{\lambda_{\max}}{\lambda_{\min}}.$$

For a general normal matrix, residual bounds for specific eigenvalue distributions are given in Proposition 2-4.3.

Proposition 2-4.3.

1. *Assume the eigenvalues of A are non-zero and contained in a disk with radius $r > 0$ and center $c \in \mathbb{C}$. Then, it can be shown that*

$$\min_{\substack{p\in\mathcal{P}_m \\ p(0)=1}} \max_k |p(\lambda_k)| \leq \left|\frac{r}{c}\right|^m$$

2. *Assume the eigenvalues of A are non-zero and contained in an ellipse with center $c \in \mathcal{R}$, focal distance $d > 0$ and major semi axis $a > 0$. Then, it can be shown that*

$$\min_{\substack{p\in\mathcal{P}_m \\ p(0)=1}} \max_k |p(\lambda_k)| \leq \left(\frac{a+\sqrt{a^2-d^2}}{c+\sqrt{c^2-d^2}}\right)^m$$

Proof. See [4, Section 6.11]. □

Proposition 2-4.4 gives a bound on the relative residual for general nonnormal matrices.

Proposition 2-4.4. *Let A be nonnormal and diagonalizable, i.e. $A = VDV^{-1}$ with $D = \text{diag}(\lambda_1, \ldots, \lambda_n)$ and V containing the eigenvectors. Then,*

$$\frac{\|r_m\|}{\|r_0\|} \leq \kappa_2(V) \min_{\substack{p\in\mathcal{P}_m \\ p(0)=1}} \max_k |p(\lambda_k)|.$$

Proof. See [46, p. 866]. □

The condition number can be computed as $\kappa_2(V) = \|V\|\|V^{-1}\|$. If the spectrum of A is contained in a circle or an ellipse, away from the origin, then the bounds in Proposition 2-4.3 can be used to simplify the min-max problem. The bound in Proposition 2-4.4 is only sharp when A is normal, though, and the aforementioned result in [45] demonstrates that it is not useful when $\kappa_2(V)$ is very large.

Recall from Equation (2-3.5) that a GMRES iteration can be expressed as $x_m = x_0 + p_{m-1}(A)r_0$. Using this identity, the m'th residual can be written as

$$\begin{aligned}\|r_m\|_2 = \|b - Ax_m\|_2 &= \|b - Ax_0 - Ap_{m-1}(A)r_0\|_2 \\ &= \|(I - Ap_{m-1}(A))r_0\|_2 \\ &= \min_{p\in\mathcal{P}_{m-1}} \|(I - Ap(A))r_0\|_2 = \min_{p\in\mathcal{P}_m, p(0)=1} \|p(A)r_0\|_2 \end{aligned} \tag{2-4.8}$$

Greenbaum and Trefethen [49] introduced the 'ideal' GMRES approximation problem, to investigate which properties of A influence the residual bound, as follows:

$$\begin{aligned}\frac{\|r_m\|}{\|r_0\|} &= \min_{\substack{p\in\mathcal{P}_m \\ p(0)=1}} \frac{\|p(A)r_0\|_2}{\|r_0\|} \\ &\leq \max_{\|v\|=1} \min_{\substack{p\in\mathcal{P}_m \\ p(0)=1}} \|p(A)v\| \\ &\leq \min_{\substack{p\in\mathcal{P}_m \\ p(0)=1}} \|p(A)\|. \end{aligned} \tag{2-4.9}$$

The last term in (2-4.9) can be bounded using the pseudospectrum [50, 51] or the field of values of A [52, 53]. Although these methods can provide insight in the effectiveness of preconditioning, the bound is again not sharp and it remains unclear how close ideal GMRES (last line of (2-4.9)) approximates worst-case GMRES (second line of (2-4.9)) [47].

2-4-4 Preconditioning

Preconditioning a Krylov subspace method can significantly improve convergence and robustness. The original system (2-3.1) is transformed into an equivalent system with the same solution, but more suitable for iterative methods in terms of, for instance, the condition number of the matrix. A preconditioner M can be applied either from the left or from the right:

$$\begin{aligned}\text{Left preconditioning}: &\quad M^{-1}Ax = M^{-1}b \\ \text{Right preconditioning}: &\quad AM^{-1}y = b,\ y = Mx\end{aligned}$$

The preconditioner should be inexpensive to construct and, ideally, a good approximation of A. $M^{-1}A$ or AM^{-1} are not explicitly formed in practice, hence the system

$Mx = z$, for arbitrary $x, z \in \mathbb{R}^n$, should be easy to solve using a direct or iterative method. In preconditioned GMRES, Arnoldi is invoked to build an orthogonal basis for the respective Krylov subspaces

$$\mathcal{K}_m(M^{-1}A, r_0) = \text{span}\{r_0, M^{-1}Ar_0, (M^{-1}A)^2 r_0, \ldots, (M^{-1}A)^{m-1} r_0\}, \text{ and}$$
$$\mathcal{K}_m(AM^{-1}, r_0) = \text{span}\{r_0, AM^{-1}r_0, (AM^{-1})^2 r_0, \ldots, (AM^{-1})^{m-1} r_0\}.$$

The pseudocode for left preconditioning is given below.

Algorithm 3 left-preconditioned restarted GMRES

1: Compute $r_0 = M^{-1}(b - Ax_0)$, $\beta = \|r_0\|_2$, and $v_1 = r_0/\beta$.
2: **for** $j = 1, 2, \ldots, m$ **do**
3: $w_j = M^{-1}Av_j$
4: **for** $i = 1, \ldots, j$ **do**
5: $h_{i,j} = (w_j, v_i)$
6: $w_j = w_j - h_{ij}v_i$
7: **end for**
8: $h_{j+1,j} = \|w_j\|_2$
9: **if** $h_{j+1,j} = 0$ **or** converged **then**
10: set $m = j$ and **go to** 14
11: **end if**
12: $v_{j+1} = w_j/h_{j+1,j}$
13: **end for**
14: Fill $\bar{H}_m = \{h_{ij}\}$ for $1 \leq i \leq m+1$, $1 \leq j \leq m$.
15: Compute the minimizer u_m of $\|\beta e_1 - \bar{H}_m u\|_2$ and set $x_m = x_0 + V_m u_m$.
16: **if** converged **then** return **else** set $x_0 = x_m$ and **go to** 1

For right preconditioning, we have $r_0 = b - Ax_0 = b - AM^{-1}y_0$. Therefore, we can take $b - Ax_0$ for the initial residual and multiply

$$y_m = y_0 + V_m u_m$$

on both sides by M^{-1}, to get

$$x_m = x_0 + M^{-1}V_m u_m.$$

The pseudocode is given in Algorithm 4.

Algorithm 4 right-preconditioned restarted GMRES

1: Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, and $v_1 = r_0/\beta$.
2: **for** $j = 1, 2, \ldots, m$ **do**
3: $w_j = AM^{-1}v_j$
4: **for** $i = 1, \ldots, j$ **do**
5: $h_{i,j} = (w_j, v_i)$
6: $w_j = w_j - h_{ij}v_i$
7: **end for**
8: $h_{j+1,j} = \|w_j\|_2$
9: **if** $h_{j+1,j} = 0$ **or** converged **then**
10: set $m = j$ and **go to** 14
11: **end if**
12: $v_{j+1} = w_j/h_{j+1,j}$
13: **end for**
14: Fill $\bar{H}_m = \{h_{ij}\}$ for $1 \leq i \leq m+1$, $1 \leq j \leq m$.
15: Compute the minimizer u_m of $\|\beta e_1 - \bar{H}_m u\|_2$ and set $x_m = x_0 + M^{-1}V_m u_m$.
16: **if** converged **then** return **else** set $x_0 = x_m$ and **go to** 1

In our applications, we focus on right preconditioning. The formulation of M will be discussed in Section 2-5. A clear advantage of right-preconditioned GMRES is that the intermediate residual norm can be obtained more easily than in left-preconditioned GMRES. When Algorithm 3 is used, any residual vector q_j computed inside the outer loop at iteration j will be preconditioned, i.e. $q_j = M^{-1}(b - Ax_j)$. To compute the intermediate residual norm, line 15 has to be used to explicitly solve for x_j, from which $\|q_j\|_2$ can be computed. In the right-preconditioned case of Algorithm 4, on the other hand, we have

$$b - AM^{-1}y_j = b - Ax_j.$$

Therefore, Proposition 2-4.2 holds, and the residual can be obtained from γ_{j+1} in the QR-decomposition of H_j.

The specific preconditioner used in IX will be introduced in the next section.

2-5 CPR

The Constraint Pressure Residual (CPR) scheme was developed by Wallis et al. as an efficient preconditioner for applications in reservoir simulation [64, 65]. The idea was further developed in a parallel environment and implemented in IX by Cao et al. [66]. The CPR method was originally motivated by the challenging nature of linear systems in reservoir simulation. Difficulties arise due to the following characteristics:

1. Strong global coupling of the variables, rendering the problems unsuitable for a straightforward parallel implementation.
2. Large and nonsymmetric systems with a varying number of unknowns per gridcell.
3. Highly heterogeneous coefficients.

Furthermore, whereas previous efforts in the reservoir simulation community were mainly directed towards solving problems on structured grids, present-day simulations often require the use of unstructured grids. Problems on complex unstructured grids, combined with the challenging factors mentioned above, result in linear systems that call for more advanced linear solvers. The preconditioners in these solvers should be sensitive to the mix of near-elliptic pressure equations and near-hyperbolic saturation equations [66, 67].

The CPR scheme employs a multi-stage preconditioner to individually tackle the numerical properties of the different variables. In particular, the pressure equations are resolved separately from the complete system. These equations exhibit an elliptic nature, which we can efficiently deal with using a multigrid preconditioner. The remaining equations are near-hyperbolic, which is efficiently solved with an Incomplete LU (ILU) preconditioner.

Remark 2-5.1. *In IX, preconditioners are based on A_{rr} (with the well terms included in the diagonal) and b_r, rather than the full Schur complement as defined in* (2-1.7)*. By doing so, we avoid computing an inverse and several matrix-matrix multiplications. The underlying assumption is that $A_{rr} \approx A_{rr} - A_{rw}A_{ww}^{-1}A_{wr}$ and $b_r \approx b_r - A_{rw}A_{ww}^{-1}b_w$. For simplicity, this distinction is omitted in the notation.*

The multigrid and ILU preconditioner will be discussed in Sections 2-5-1 and 2-5-2, respectively. For an introduction to matrix splitting and basic iterative methods, we refer to [4]. Assuming that both preconditioners can be expressed in matrix form, the two subsequent solution corrections to proceed from iteration i to iteration $i+1$ in a two-stage preconditioner can be expressed as

$$\begin{aligned} x_{i+\frac{1}{2}} &= x_i + d_1 = x_i + M_1^{-1}(b - Ax_i), \\ x_{i+1} &= x_{i+\frac{1}{2}} + d_2 = x_{i+\frac{1}{2}} + M_2^{-1}(b - Ax_{i+\frac{1}{2}}). \end{aligned} \tag{2-5.1}$$

Elementary choices include $M = A$ (convergence in one step, expensive) and $M = I$ (slow convergence, cheap). By weighing speed and computational costs, we often seek a

balance between these two choices. Combining the identities in (2-5.1) results in

$$x_{i+1} = x_i + M_{\text{CPR}}^{-1}(b - Ax_i),$$

where

$$\begin{aligned} M_{\text{CPR}}^{-1} &= M_1^{-1} + M_2^{-1} - M_2^{-1}AM_1^{-1} \\ &= M_1^{-1} + M_2^{-1}(I - AM_1^{-1}). \end{aligned}$$

In the CPR scheme, M_2^{-1} represents the stage two ILU preconditioner denoted as M_{ILU}^{-1}. M_1^{-1} corresponds to stage one and is chosen as

$$M_1^{-1} = C(C^T \text{diag}^{-1}(A)AC)^{-1}C^T,$$

where $\text{diag}(A) \in \mathbb{R}^{n \times n}$ holds the block diagonal of A and, assuming the pressure variable is the last unknown in each cell, C is the $n \times n_c$ matrix[6]

$$C = \begin{bmatrix} e_{\text{pres}} & & & \\ & e_{\text{pres}} & & \\ & & \ddots & \\ & & & e_{\text{pres}} \end{bmatrix}, \qquad e_{\text{pres}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \in \mathbb{R}^{n_u \times 1}, \tag{2-5.2}$$

restricting the linear system to the pressure equations. The expression for $M_{\text{CPR}}^{-1}r_i$ thus becomes

$$M_{\text{CPR}}^{-1}r_i = M_{\text{ILU}}^{-1}(r_i - AC(C^T \text{diag}^{-1}(A)AC)^{-1}C^T r_i) \tag{2-5.3}$$

$$+C(C^T \text{diag}^{-1}(A)AC)^{-1}C^T r_i. \tag{2-5.4}$$

In the actual implementation, M_1^{-1} and M_2^{-1} are not computed explicitly. Instead, the following procedure is followed to evaluate (2-5.3) [66]:

1. Restrict the full system residual to the pressure residual, i.e.

$$r_i^{\text{pres}} = C^T r_i.$$

2. Compute the stage one (pressure) solution correction d_1^{pres} in (2-5.1) by solving

$$(C^T \text{diag}^{-1}(A)AC)d_1^{\text{pres}} = r_i.$$

3. Expand d_1 to the full system, i.e.

$$d_1 = Cd_1^{\text{pres}}.$$

[6] Recall that we assumed that there are n_c cells in the computational grid.

4. Correct the full system residual with the pressure solution correction, i.e.

$$r_i^{\text{corr}} = r_i - Ad_1.$$

5. Compute the stage two solution correction d_2 in (2-5.1) by solving

$$M_{\text{ILU}} d_2 = r_i^{\text{corr}}.$$

Having obtained d_1 and d_2, the next iteration is $x_{i+1} = x_i + M_{\text{CPR}}^{-1} r_i = x_i + d_2 + d_1$.

Stage one (the pressure solve) of the CPR preconditioner eliminates the low-frequency errors. Stage two, which incorporates the result of stage one in a residual correction, tackles the high-frequency errors. Equivalently, stage one works on the long-range error components of the pressure field, whereas stage two is applied to variables with a local dependency. The motivation for this idea is that dealing with long-range dependency is computationally more demanding. By applying the multigrid preconditioner to a smaller linear (pressure) system, the computational cost is reduced and the error characteristics are effectively handled. In addition, algebraic multigrid with a single V-cycle is not effective when applied to hyperbolic systems.

2-5-1 CPR stage 1: AMG

Algebraic multigrid (AMG) is a multigrid method [69, 70]. For a detailed introduction, we refer to [68, pp. 413-532]. The error is first smoothed on the fine grid, after which the result is restricted to the coarse grid with a restriction operator. On the coarse grid, the slowly varying error modes are resolved to obtain a solution correction, which is prolongated back to the fine grid with a prolongation operator. In contrast with geometric multigrid, coarsening and interpolation operators in AMG are determined algebraically using only the coefficients of the fine-grid matrix. AMG follows a similar approach taken in the Black Box Multigrid Method (BoxMG) [71], in which the coefficients of the interpolation operator are derived from the connectivity graph of each gridpoint and the corresponding coefficient values.

The advantage of AMG is that the method, in contrast to BoxMG, can also be used for unstructured grids. It is notoriously difficult to apply multigrid to non-rectangular grids, especially in three dimensions [4]. Furthermore, geometric multigrid does not efficiently account for heterogeneous and anisotropic coefficients. By construction, AMG is more robust for these types of problems. The increased robustness comes at a cost, though, in the following disadvantages of AMG:

- High set up cost
- Unstructured grid hierarchy
- Difficult to parallelize

Regardless, because of the increased robustness and black box nature of the method, AMG has gained widespread popularity in commercial codes [12].

The next lemma relates a property of the error to the coefficients a_{ij} of A, when A is positive definite.

Lemma 2-5.1. *For a smooth error vector s and A positive definite, we have*

$$\sum_{j \neq i} \frac{|a_{ij}|}{a_{ii}} \left(\frac{s_i - s_j}{s_i} \right)^2 \ll 1 \tag{2-5.5}$$

Proof. See e.g. [4, p.457] or [68, p.439]. □

Note that A is not positive definite in our case. No comparable theory exists for non-symmetric matrices, but practical experience indicates that other other properties of the matrix, such as a lack of (weak) diagonal dominance, influence the performance of AMG more than a lack of symmetry [68].

Remark 2-5.2. *An ambiguity exists regarding the term 'smooth'. In geometric multigrid, a smooth error is interpreted as being smooth relative to the coarser level. That is, we say the error is smooth when the the result of the smoothing is slowly-varying on the predefined coarse level. In algebraic multigrid, on the other hand, there is no predefined coarse level. Hence, a 'smooth' error in AMG is defined as being smooth relative to the predefined smoother. We say the error e is algebraically smooth when the smoother S has little effect, i.e. $Se \approx e$. Very small eigenvalues, as we will see in Chapter 3, generally slow down the convergence of the smoother. Therefore, these eigenvalues correspond to algebraically smooth error. More details on this topic can be found in [68].*

From Lemma 2-5.1 we derive that for (2-5.5) to hold, $(s_i - s_j)/s_i$ should be small when $|a_{ij}|/a_{ii}$ is large. Equivalently, the error s is slowly-varying in the direction of the connection between node i and node j if the corresponding weight is relatively large. The latter idea is captured in Definition 2-5.1.

Definition 2-5.1. *A connection of a coarse node i to an adjacent node j, i.e. $a_{ij} \neq 0$, is said to be 'strong' when $|a_{ij}|/a_{ii}$ is larger than a certain threshold. If the fraction is smaller than the threshold, then the connection is said to be 'weak'.*

A coarse grid is formed by taking a maximal independent set of the strong connections. To account for the many variations on the process of identifying strong connections (for different classes of matrices), a general AMG framework has been developed in [72]. The auxiliary matrix P is defined as the 'primary' matrix and is used (instead of A) to establish the set of strong connections [73].

In general, AMG methods fall into one of the following three categories [72]:

1. In classical, **Variable-based AMG** [68], we simply take $P = A$. This approach does not account for the scenario in which multiple unknowns (pressure and saturation, for example) are defined on the same gridpoints. Hence, variable-based AMG only works for scalar PDEs with a single unknown.

2. In **Unknown-based AMG** [69], the linear system $Ax = b$ is reordered 'unknown-wise', as

$$\begin{bmatrix} A_{[1,1]} & \dots & A_{[1,n_u]} \\ \vdots & \ddots & \vdots \\ A_{[n_u,1]} & \dots & A_{[n_u,n_u]} \end{bmatrix} = \begin{bmatrix} x_{[1]} \\ \vdots \\ x_{[n_u]} \end{bmatrix} = \begin{bmatrix} b_{[1]} \\ \vdots \\ b_{[n_u]} \end{bmatrix},$$

where n_u denotes the number of unknowns per gridcell. Each submatrix $A_{[i,j]}$, $1 \leq i, j \leq n_u$, $i \neq j$, corresponds to the coupling between two unknowns. The primary matrix P is then defined by ignoring the cross-coupling:

$$P = \begin{bmatrix} A[1,1] & & \\ & \ddots & \\ & & A[n_u, n_u] \end{bmatrix}.$$

In unknown-based AMG, variable-based AMG is applied to each diagonal block in P. Consequently, coarsening and interpolation are performed separately for each unknown. The advantage of this approach is that distinct heterogeneity in different unknowns can be efficiently dealt with. If error characteristics of different unknowns are connected, however, ignoring strong cross-coupling is ineffective.

3. In **Point-based AMG** [69, 76], the linear system $Ax = b$ is reordered 'point-wise', as

$$\begin{bmatrix} A_{[1,1]} & \dots & A_{[1,n_c]} \\ \vdots & \ddots & \vdots \\ A_{[n_c,1]} & \dots & A_{[n_c,n_c]} \end{bmatrix} = \begin{bmatrix} x_{[1]} \\ \vdots \\ x_{[n_c]} \end{bmatrix} = \begin{bmatrix} b_{[1]} \\ \vdots \\ b_{[n_c]} \end{bmatrix},$$

where n_c denotes the number of cells in the grid. In this case, each submatrix $A_{[ij]}$, $1 \leq i, j \leq n_c$, $i \neq j$, represents the coupling between two gridpoints. Different unknowns are required to be discretized on the same grid (i.e. no staggered grids), but not all unknowns have to be defined on each gridpoint. Two possibilities for entries p_{ij} of P in this method are the block approaches [72]

$$\begin{aligned} &p_{ij} = -\|A_{[i,j]}\| \text{ and } p_{ii} = \|A_{[i,i]}\|, \text{ or} \\ &p_{ij} = -\|A_{[i,j]}\| \text{ and } p_{ii} = -\sum_{i \neq j} p_{ij}. \end{aligned}$$

Alternatively, entries of P can be based on distances between points, which is closely related to geometric multigrid.

Having established the coarsening operator using the primary matrix, there are various ways to construct the interpolation operator, such as:

- Separate for each unknown, i.e. using the blocks in an unknown-wise reordering of A.
- Equivalent for each unknown, using the blocks in a point-wise reordering of A
- Equivalent for each unknown, based on the coordinates.

Finding the optimal combination of a coarsening approach and an interpolation scheme for different applications is the subject of continuous research [74].

In IX, the implementation of AMG is closely related to the unknown-based approach. Using the restriction matrix C from (2-5.2), the full system is restricted to the pressure system. AMG is applied to the primary matrix

$$P = C^T \text{diag}^{-1}(A) AC.$$

Interpolation is based on the values in P. Although AMG can be used as a solver, it is used as a preconditioner for GMRES in IX. This process, which is referred to as 'acceleration' of AMG, is generally more robust [68, 75]. A single step of GMRES is preconditioned by a V-cycle of AMG. The default amount of coarsening steps is twenty.

To illustrate coarsening in IX, a 81×2 sideview of a reservoir is shown in Figure 2-5.1. Nodes with colors towards the right of the colorbar are included more frequently on coarser levels. Some nodes, which are not shown at all, are never included. Observe that nodes around the well, where flow occurs, are included more often.

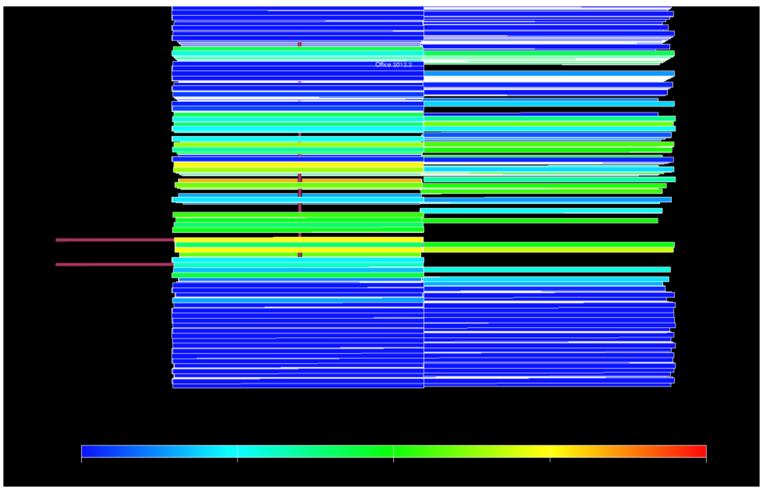


Figure 2-5.1: Coarsening of AMG. Colors towards the right of the colorbar imply more frequent inclusion of the node on coarser levels.

2-5-2 CPR stage 2: ILU

For a detailed discussion of Incomplete LU factorization, we refer to [4, 77]. The LU decomposition can be used as a direct solver by computing the upper triangular matrix U and the lower triangular matrix L such that $A = LU$. $Ax = b$ is then easily solved by doing a forward substitution in $Ly = b$ followed by a backward substitution in $Ux = y$. For symmetric positive definite matrices, the Cholesky decomposition follows a similar approach using $A = LL^T$.

Due to the fill in, performing an LU decomposition on a large, sparse matrix is expensive. For $A \in \mathbb{R}^{n \times n}$, approximately $\mathcal{O}(n^2)$ flops are required for an LU solve, which is not competitive with iterative solvers [78]. Hence, in practice the LU method is often used in incomplete form as a preconditioner for an iterative solver. We apply the LU decomposition such that

$$R = A - LU \tag{2-5.6}$$

satisfies certain constraints, such as a predefined zero pattern. In ILU(0), for example, we require

$$\begin{cases} (LU)_{ij} = a_{ij} & \text{for } a_{ij} \neq 0 \\ (LU)_{ij} = 0 & \text{for } a_{ij} = 0 \end{cases} \tag{2-5.7}$$

where a_{ij} and $(LU)_{ij}$ are the coefficients of A and LU, respectively. To satisfy (2-5.6), equation (2-5.7) implies $r_{ij} = a_{ij}$ for $a_{ij} = 0$. ILU(0) is referred to as 'zero-order fill in'. To obtain first-order fill in, ILU(1), ILU(0) is applied a second time, using the zero pattern from the first ILU(0) step. This process can be repeated p times to obtain the ILU(p) factorization.

Instead of treating A point-wise, ILU can also be applied in a block-wise approach, i.e.

$$\begin{cases} (LU)_{[i,j]} = A_{[i,j]} & \text{for } A_{[i,j]} \neq 0 \\ (LU)_{[i,j]} = 0 & \text{for } A_{[i,j]} = 0 \end{cases} \tag{2-5.8}$$

Since this method is more efficient, IX contains a block-wise ILU factorization.

The advantages of ILU methods are the simplicity and the robustness for a broad class of problems. The main drawback of ILU is the lack of scalability. The algorithm is inherently sequential, because the LU factorization is executed column by column. Furthermore, applying the ILU preconditioner requires a forward and backward solve, which are also sequential. The optimal use of ILU in the second stage of the CPR preconditioner is subject of further research. Depending on the application, a balance is sought between the cheap but inaccurate ILU(0) method and the more expensive but also more accurate ILU(1) method. Multicoloring can be used to parallelize the sequential steps in the factorization. In most of the cases under consideration in Chapter 5, ILU(1) is used as a preconditioner to single iteration of GMRES. For more details, see e.g. [75].

Chapter 3

Deflation

3-1 Motivation

As a motivation for deflation methods, consider the following example of the convergence of GMRES.

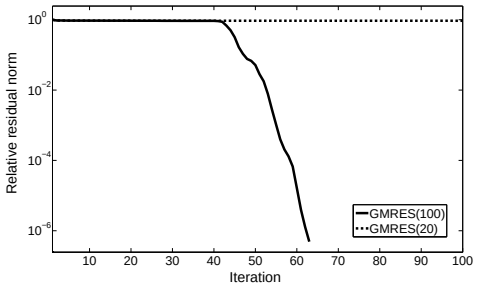
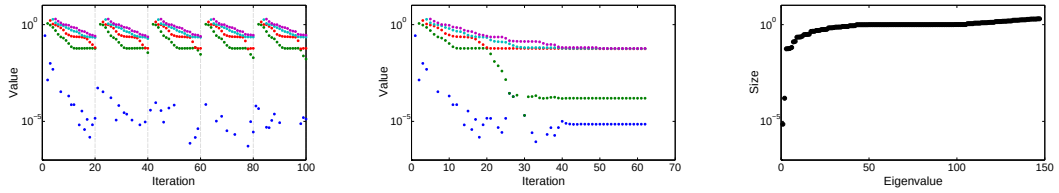


Figure 3-1.1: GMRES example with tolerance 1e-6.

Full GMRES corresponds to Algorithm 4 with a sufficiently large m to prevent restarting. In this case, $m = 100$. The dashed line results from choosing $m = 20$. GMRES(20) does not convergence, whereas GMRES(100) converges only aftert about 40 iterations. A major drawback of full GMRES, in this case GMRES(100), is the fact that the computational costs increase quadratically with the number of iterations. As the matrices $\bar{H}_m$ and V_m grow, solving (2-4.3) becomes increasingly more difficult. Hence, in practice, full GMRES is seldom used. The example shows, however, that restarted GMRES might not converge. Hence, explaining the difference between the two lines in Figure 3-1.1 is crucial.

The superlinear convergence of GMRES was associated by Van der Vorst and Vuik [7] with the convergence of the Ritz values of $\bar{H}_m$ to the eigenvalues of the operator A. If the Krylov subspace reaches a sufficient size, the Ritz values will be close to the eigenvalues of A. From that point on, GMRES will behave as if these approximated eigenvalues have been *naturally deflated* from A, resulting in faster convergence. To illustrate this idea, the Ritz values corresponding to the convergence history in Figure 3-1.1 are plotted below.



(a) (b) (c)

Figure 3-1.2: Smallest Ritz values for (a) GMRES(20) and (b) GMRES(100), and the spectrum of A (c).

In Figure 3-1.2(a), the five smallest Ritz values of the restarted GMRES case in Figure 3-1.1 are plotted. Clearly, the convergence of the Ritz values towards the exact eigenvalues is reset after each restart. In Figure 3-1.2(b), GMRES is not restarted, and the two smallest Ritz values convergence to the two smallest eigenvalues in Figure 3-1.2(c). Comparing this result to Figure 3-1.1 shows that precisely as the Ritz values approach the exact eigenvalues, at about 40 iterations, convergence of GMRES becomes superlinear.

Natural deflation occurs, because eigenvector components corresponding to eigenvalues are removed from the linear system. Extreme eigenvalues, in particular, are detrimental to the converge of GMRES [8, 9, 10]. Unfortunately, a restart in GMRES erases the obtained Krylov subspace before it might reach sufficient size to allow for natural deflation, as illustrated in Figure 3-1.2. In addition, even if the Krylov subspace grows big enough and deflation occurs, it will only aid to convergence in the current cycle. This discussion suggests convergence would improve if (small) eigenvalues could be removed *artificially*. This idea gives rise to the method of deflation.

3-2 Overview

After the introduction of deflation by Nicolaides [23] and Dostál [24], deflation was further developed for symmetric and non-symmetric linear systems in a wide range of applications. Deflation algorithms differ mainly in the way the deflation operator is constructed, and the way it is applied. As for the application, two general approaches can be distinguished:

1. **Augmented subspace deflation.** In a series of papers, Morgan [17, 54, 55], and Chapman and Saad [16] introduced the idea to apply deflation after a restart of GMRES. At the end of a GMRES cycle, augmented subspace deflation methods retain information from the generated Krylov subspace. Information is added to the search space of the next cycle, in order to directly reveal knowledge about the search space that the Krylov subspace method itself would take much longer to find. Instead of starting fresh, for example, approximated eigenvectors corresponding to the smallest eigenvalues of A can be selected to augment the search space of the next cycle.

2. **Deflation by preconditioning.** In contrast to augmented subspace approaches, preconditioning deflation methods use approximated eigenvectors to construct a preconditioner; see for example [9, 10, 56, 57, 63]. This preconditioner can be frozen for all subsequent cycles, or refreshed when necessary. In some work, harmful eigenvalues are projected to zero, while others have proposed to move extreme parts of the spectrum to the vicinity of the largest eigenvalue. In projection preconditioning, the harmful eigenvector components are explicitly deflated by a projection, after which the Krylov subspace method is used to solve the deflated linear system.

In dynamic deflation, the information generated during a GMRES cycle is employed to construct the deflation vectors. After each restart, the deflation subspace can be updated. An alternative approach is static deflation, which uses a fixed set of deflation vectors that are available a priori. Physical information, such as the properties of shale and sandstone layers underground, can be exploited to construct the deflation subspace [11]. The latter approach is part of a technique known as domain-based deflation, which will be further discussed in Section 3-4. Because static deflation requires the deflation operator to be computed only once, we prefer this approach for reasons of computational efficiency.

Simplicity plays an important role in commercial software packages such as IX. Hence, a black-box deflation method is favored over algorithms that require the user to tune certain parameters. Although to a certain extent user input will always be necessary for efficient deflation, we believe deflation by preconditioning is easier to use, and easier to implement, than augmented subspace deflation. A deflation preconditioner can simply be incorporated in the existing CPR, AMG or ILU operator, to obtain a deflated two-level preconditioned algorithm. Due to the large cases used in IX, explicitly building and multiplying a preconditioner with the size of A is undesirable. As will become clear in the next section, we can avoid this by using projection preconditioning.

Deflation can be applied to either the full (reservoir) system with all unknowns, or the pressure system only. In the full system, deflation will form a two-level preconditioning method with the ILU preconditioner, while in the pressure system deflation is combined with AMG. The interplay of the preconditioner and deflation will influence our experiments, as we will demonstrate in Section 5.

In conclusion, our focus will be on static deflation methods using a projection preconditioner, applied to both the full (non-symmetric) system and the pressure system. In the upcoming sections, the corresponding mathematical framework will be presented. In particular, we will discuss a number of approaches to compute the deflation vectors in Section 3-4.

3-3 Framework

For a thorough introduction of deflation methods for symmetric problems, we refer to [12]. We will follow the derivation for the non-symmetric case, discussed in [8].

Definition 3-3.1. *Let $A \in \mathbb{R}^{n\times n}$ be a non-symmetric matrix as given in* (2-3.1)*, and assume that the deflation matrix $Z \in \mathbb{R}^{n\times d}$ is given. Then we define the matrix $E \in \mathbb{R}^{d\times d}$ as*

$$E = Z^T AZ, \tag{3-3.1}$$

and the deflation matrices P_1 and P_2 as

$$P_1 = I - AZE^{-1}Z^T,$$
$$P_2 = I - ZE^{-1}Z^T A.$$

Remark 3-3.1.

- *Inverting E is relatively cheap, since, in general, $d << n$.*
- *In the symmetric case we have $P_1^T = P_2$.*

3-3-1 Galerkin matrix

E is essentially the result of a prolongation-restriction operation, known in multigrid terminology as a Galerkin matrix. Indeed, deflation and multigrid are related. The columns of Z should approximate slow-varying error modes, which are often determined by eigenvector components corresponding to small eigenvalues. Z^T and Z are the restriction operator and the projection operator, respectively, and P_1 and P_2 can be interpreted as coarse-grid corrections. In the symmetric case, these coarse-grid corrections are of the form

$$P = I + ZE^{-1}Z^T. \tag{3-3.2}$$

Nabben and Vuik show, however, that a symmetric linear system preconditioned by 3-3.2 always has a larger effective condition number than the deflated preconditioned system [58]. Hence, the latter can be expected to show faster convergence when the CG method is used.

If A is (symmetric) positive definite, then, as we show the next lemma, E is nonsingular.

Lemma 3-3.1 (cf. [12, Lemma 3.1] and [8, Lemma 5.2])**.** *Let Z and E be given as in Definition 3-3.1, and assume A is positive definite and Z has full rank. Then, E is nonsingular.*

Proof. Because every positive definite matrix is invertible, it is sufficient to show that E is positive definite. For A positive definite, we have

$$y^T A y > 0 \qquad \forall\ y \in \mathbb{R}^n,\ y \neq 0.$$

In particular, let $y = Z\tilde{y}$, where $\tilde{y}$ satisfies the same requirements as y, then

$$\tilde{y}^T E \tilde{y} = \tilde{y}^T Z^T A Z \tilde{y} = (Z\tilde{y})^T A (Z\tilde{y}) = y^T A y > 0, \qquad \forall\ \tilde{y} \in \mathbb{R}^n,\ \tilde{y} \neq 0,$$

as required. □

In our case, however, A cannot be guaranteed to be symmetric or positive (semi-)definite. Fortunately, a proof that E is nonsingular can also be given under the assumption that the columns of Z form a basis for an A-invariant subspace. A trivial example of an A-invariant subspace is $Z = [v_1, \ldots, v_d], 1 \leq d \leq n$, where $v_1, \ldots, v_n$ are the exact eigenvectors of A.

Lemma 3-3.2 (cf. [15, Proposition 3.2])**.** *Let Z and E be given as in Definition 3-3.1, and assume A is nonsingular and the columns of Z form a basis for an A-invariant subspace $\mathcal{Z}$. In addition, let $W \in \mathbb{R}^{n\times(n-d)}$ be the matrix whose columns form a basis for $\mathcal{Z}^\perp$. Then, E is nonsingular, and A can be decomposed as*

$$A = [Z, W] B [Z, W]^{-1}, \text{ with } B = \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix} \tag{3-3.3}$$

for some $B_{11} \in \mathbb{R}^{d\times d}$, $B_{12} \in \mathbb{R}^{d\times(n-d)}$ and $B_{22} \in \mathbb{R}^{(n-d)\times(n-d)}$.

Proof. First, note that the decomposition of A follows from

$$A[Z, W] = [AZ, AW] \tag{3-3.4}$$
$$= [ZB_{11}, B_{12}Z + B_{22}W] \tag{3-3.5}$$
$$= [Z, W] \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix}, \tag{3-3.6}$$

for some B_{11}, B_{12} and B_{22} given as given above. We used that Z is an A-invariant subspace, i.e. $AZ = ZB_{11}$, which can also be used to write

$$E = Z^T A Z = Z^T Z B_{11}. \tag{3-3.7}$$

As long as Z is a basis, the columns are linearly independent and $Z^T Z$ will be nonsingular. In addition, $\det(A) = \det(B_{11})\det(B_{22})$ and A is nonsingular, hence B_{11} is nonsingular. Combining the latter two conclusions in (3-3.7) leads to the nonsingularity of E. □

3-3-2 Deflated system

Before deriving the deflated system, we first prove a number of useful properties.

Lemma 3-3.3. *Let A, E, P_1 and P_2 be given as in Definition 3-3.1. Then, the following equalities hold:*

(a) $P_1^2 = P_1$ *and* $P_2^2 = P_2$*;*

(b) $P_1 A = A P_2$*;*

(c) $P_1 A Z = 0$*;*

(d) $P_2 Z = 0$*;*

(e) $(I - P_2)x = Z E^{-1} Z^T b$.

Proof.

(a) $P_1^2 = (I - AZE^{-1}Z^T)^2 = I - 2AZE^{-1}Z^T + AZE^{-1}(Z^TAZ)E^{-1}Z^T$
$= I - 2AZE^{-1}Z^T + AZE^{-1}Z^T = P_1$, and
$P_2^2 = (I - ZE^{-1}Z^TA)^2 = I - 2ZE^{-1}Z^TA + ZE^{-1}(Z^TAZ)E^{-1}Z^TA$
$= I - 2ZE^{-1}Z^TA + ZE^{-1}Z^TA = P_2$;

(b) $P_1 A = (I - AZE^{-1}Z^T)A = A - AZE^{-1}Z^TA = A(I - ZE^{-1}Z^TA) = AP_2$;

(c) $P_1 AZ = (I - AZE^{-1}Z^T)AZ = AZ - AZ(Z^TAZ)^{-1}Z^TAZ = AZ - AZ = 0$;

(d) $P_2 Z = (I - ZE^{-1}Z^TA)Z = Z - Z(Z^TAZ)^{-1}Z^TAZ = Z - Z = 0$;

(e) $(I - P_2)x = ZE^{-1}Z^TAx = ZE^{-1}Z^Tb$.

□

Corollary 3-3.1. *Let A, E, P_1 and P_2 be given as in Definition 3-3.1. Then, $P_1 A$ has d zero eigenvalues.*

Proof. Lemma 3-3.3(c) implies that each column in Z corresponds to one zero eigenvalue, i.e. Z is the nullspace of $P_1 A$. □

Following the presentation in [8], we split the solution x to (2-3.1) in two parts:

$$x = (I - P_2)x + P_2 x. \tag{3-3.8}$$

The first part can be simplified using Lemma 3-3.3e, which eliminates dependency on x. Substituting this result in (3-3.8) and left-multiplication by A yields

$$\begin{aligned} Ax = AZE^{-1}Z^Tb + AP_2x \;&\Leftrightarrow\; b = AZE^{-1}Z^Tb + P_1Ax \\ &\Leftrightarrow\; (I - AZE^{-1}Z^T)b = P_1Ax \\ &\Leftrightarrow\; P_1 b = P_1 Ax, \end{aligned} \tag{3-3.9}$$

where we also used Lemma 3-3.3b. Note that P_1A has at least one zero eigenvalue, by Corallary 3-3.1, so the system is singular. Consequently, the solution x is not necessarily the solution of the original linear system $Ax = b$, as x may contain components in the nullspace of P_1A. Rather, we refer to 3-3.9 as the 'deflated system', and define $\hat{x}$ as its solution, i.e.

$$P_1A\hat{x} = P_1b. \tag{3-3.10}$$

The following proposition relates the deflated solution back to the original solution.

Proposition 3-3.1 (cf. [12, Lemma 3.5])**.** *Let x and $\hat{x}$ be the solutions of* (2-3.1) *and* (3-3.10)*, respectively, and let P_2 and Z be given as in Definition 3-3.1. Then,*

$$P_2\hat{x} = P_2x.$$

Proof. Decompose the solution to the deflated system as $\hat{x} = x + z$, with $z \in \mathcal{N}(P_1A) = \mathcal{R}(Z)$ (Lemma 3-3.3c). Using Lemma 3-3.3d, we find

$$P_2\hat{x} = P_2x + P_2z = P_2x, \tag{3-3.11}$$

as required. □

Hence, even though (3-3.10) is singular, the projected solution $P_2\hat{x}$ is unique because it has no components in the null space $\mathcal{N}(P_1A)$.

In conclusion, the solution to the original linear system can be found by solving (3-3.10) for $\hat{x}$, and substituting the result in

$$x = ZE^{-1}Z^Tb + P_2\hat{x}. \tag{3-3.12}$$

In the upcoming sections, we will use the notation $x^* = ZE^{-1}Z^Tb$.

3-3-3 Geometric illustration

An intuitive two-dimensional geometric illustration of the splitting $\hat{x} = x + z$ is given in Figure 3-3.1. We assume $\lambda_1 << \lambda_2$, with corresponding eigenvectors v_1 and v_2.

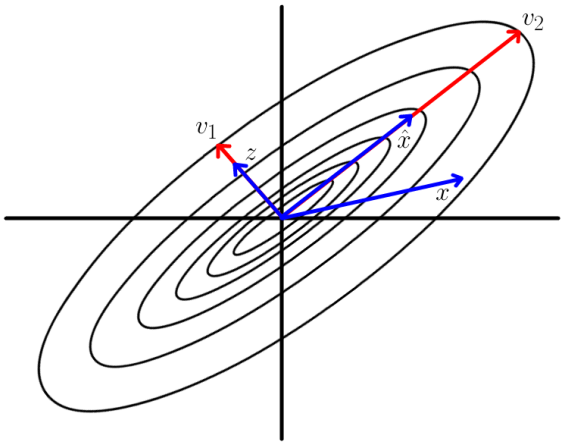


Figure 3-3.1: 2D illustration of deflation.

Taking the eigenvector corresponding to the small eigenvalue as the deflation matrix Z, we get $v_1 = \mathcal{R}(Z)$ and $v_2 = \mathcal{R}^{\perp}(Z)$. The action of the projector P_2 then becomes

$$\begin{cases} P_2 z = 0 \\ P_2 x = \hat{x}, \end{cases}$$

which leads to the result of (3-3.11). By projecting on the subspace perpendicular to $\mathcal{R}(Z)$, the eigenvector component corresponding to the small eigenvalue λ_1 is omitted in the deflated system.

3-3-4 Convergence

To discuss the effect deflation has on the convergence of GMRES, we first analyze the spectrum. Corollary 3-3.2 is an extension of Corollary 3-3.1.

Corollary 3-3.2 (cf. [15, Proposition 3.2 and Corollary 3.3])**.** *Let A and P_1 be given as in Definition 3-3.1, and consider* (3-3.3) *under the corresponding assumption of Lemma 3-3.2. Then, $\sigma(A) = \sigma(B_{11}) \cup \sigma(B_{22})$ in the original linear system* (2-3.1)*, and $\sigma(P_1 A) = \{0, \ldots, 0\} \cup \sigma(B_{22})$ in the deflated system* (3-3.10)*.*

Proof. The spectrum of A follows directly from Equation (3-3.3). Furthermore, recall from Lemma 3-3.2 that $AZ = ZB_{11}$, $AW = ZB_{12} + WB_{22}$ and $Z^T W = 0$. As a result,

we find

$$\begin{aligned} P_1AW &= (I - AZE^{-1}Z^T)AW \\ &= AW - AZ(Z^TAZ)^{-1}Z^TAW \\ &= AW - AZ(Z^TAZ)^{-1}Z^T(ZB_{12} + WB_{22}) \\ &= AW - AZ(Z^TAZ)^{-1}Z^TZB_{12} \\ &= AW - ZB_{11}(Z^TZB_{11})^{-1}Z^TZB_{12} \\ &= AW - ZB_{11}B_{11}^{-1}Z^{-1}Z^{-T}Z^TZB_{12} \\ &= AW - ZB_{12} \\ &= WB_{22}. \end{aligned}$$

In addition, by Lemma 3-3.3(c), $P_1AZ = 0$. In conclusion,

$$P_1A[Z, W] = [0, WB_{22}] = [Z, W]\begin{bmatrix} 0 & 0 \\ 0 & B_{22} \end{bmatrix},$$

from which we derive the spectrum $\sigma(P_1A) = \{0, \ldots, 0\} \cup \sigma(B_{22})$. □

The Corollary shows that in the deflated system the first d (sorted) eigenvalues are projected to 0, while the remaining $n-d$ eigenvalues remain untouched. In the symmetric case, this would be sufficient to show that the condition number of the deflated system is better than the condition number of the original linear system [8, Theorem 2.2]. In the nonsymmetric case, this result does not hold, although similar convergence behavior was observed by the authors of [8] as long as the asymmetric part of A is not too dominant.

While a proof involving the condition number cannot be given for nonsymmetric A, we can show that the residuals in deflated GMRES are smaller than or equal to the residual in regular GMRES, as long as Z contains a basis for some A-invariant subspace [15]. We begin by carrying out a QR decomposition on $[Z, \tilde{Z}] = [z_1, \ldots, z_d, z_{d+1}, \ldots, z_n]$, where $z_1, \ldots, z_n$ are the deflation vectors, and $z_{d+1}, \ldots, z_n$ are appended to form a basis of $\mathbb{R}^n$:

$$[Z, \tilde{Z}] = QR = [Q_1, Q_2]\begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}, \tag{3-3.13}$$

where $Q_1 \in \mathbb{R}^{n\times d}$, $Q_2 \in \mathbb{R}^{n\times(n-d)}$, $R_{11} \in \mathbb{R}^{d\times d}$, $R_{12} \in \mathbb{R}^{d\times(n-d)}$ and $R_{22} \in \mathbb{R}^{(n-d)\times(n-d)}$. We can then express P and PA in terms of elements of (3-3.3) and (3-3.13).

Lemma 3-3.4. *Let P_1 and A be given as in Definition 3-3.1 and Lemma 3-3.2. We then have*

(a) $P_1 = Q_2Q_2^T$;

(b) $P_1 A = Q_2 B_2 2 Q_2^T$;

(c) $P_1 A P_1 = P_1 A$.

Proof. See [15, Lemma 4.1]. □

Lemma 3-3.4 can be used to prove the following theorem regarding the convergence of deflated GMRES.

Theorem 3-3.1 (cf. [15, Theorem 5.1])**.** *Assume the columns of the deflation matrix Z form a basis of some A-invariant subspace. Let r_m and $\hat{r}_m$ be the m'th residual of the original linear system* (2-3.1) *and the deflated system* (3-3.10)*, respectively, solved using GMRES. Then, starting with the same initial guess, we have*

$$\|\hat{r}_m\|_2 \leq \|r_m\|_2 \qquad \forall\ m = 1, 2, \ldots$$

Proof. The proof consists of elements throughout the paper [15], but will be given here for completeness. Let $x_0 \in \mathbb{R}^n$ be the initial guess, and $r_0 = b - Ax_0$ the corresponding intial residual. Recall from (2-4.8),

$$\|r_m\|_2 = \min_{p \in \mathcal{P}_m, p(0)=1} \|p(A) r_0\|_2.$$

The initial residual of the deflated system satisfies the relation $\hat{r}_0 = P_1 b - P_1 A x_0 = P_1 r_0$. Let $p_m(B) = 1 + \sum_{k=1}^m c_k B^k$ for any matrix B and scalars $c_1, c_2, \ldots, c_m$, then,

$$\begin{aligned} p_m(P_1 A)\hat{r}_0 = p_m(P_1 A) P_1 r_0 &= P_1 r_0 + \sum_{k=1}^m c_k (P_1 A)^k P_1 r_0 \\ &= P_1 r_0 + \sum_{k=1}^m c_k P_1 A^k r_0 \\ &= P_1 (r_0 + \sum_{k=1}^m c_k A^k r_0 = P_1 p_m(A) r_0, \end{aligned} \tag{3-3.14}$$

where we used Lemma 3-3.4(c). Combining results yields

$$\begin{aligned} \|\hat{r}_m\|_2 = \min_{p \in \mathcal{P}_m, p(0)=1} \|p(P_1 A)\hat{r}_0\|_2 &\leq \|p_m(P_1 A)\hat{r}_0\|_2 && \text{by (2-4.8)}, \\ &= \|P_1 p_m(A) r_0\|_2 && \text{by (3-3.14)}, \\ &= \|Q_2 Q_2^T p_m(A) r_0\|_2 && \text{by Lemma } 3-3.4\text{(a)}, \\ &= \|Q_2^T p_m(A) r_0\|_2 && \text{by orthogonality}, \end{aligned}$$

for all $p_m \in \mathcal{P}_m$ with $p_m(0) = 1$. Hence, we found

$$\|\hat{r}_m\|_2 \leq \|Q_2^T p_m(A) r_0\|_2. \tag{3-3.15}$$

Next, note that

$$p_m(A)r_0 = QQ^T p_m(A)r_0 = Q_1Q_1^T p_m(A)r_0 + Q_2Q_2^T p_m(A)r_0,$$

since $QQ^T = I$. Also by orthogonality, $(Q_1Q_1^T p_m(A)r_0)^T(Q_2Q_2^T p_m(A)r_0) = 0$, and hence Pythagoras gives

$$\|p_m(A)r_0\|_2^2 = \|Q_1^T p_m(A)r_0\|_2^2 + \|Q_2^T p_m(A)r_0\|_2^2.$$

The deflated residual in Equation (3-3.15) can now be written as

$$\|\hat{r}_m\|_2^2 \leq \|Q_2^T p_m(A)r_0\|_2^2 = \|p_m(A)r_0\|_2^2 - \|Q_1^T p_m(A)r_0\|_2^2.$$

In particular, let $p_m(A)r_0 = r_m$, then

$$\|\hat{r}_m\|_2^2 \leq \|Q_2^T p_m(A)r_0\|_2^2 = \|r_m\|_2^2 - \|Q_1^T r_m\|_2^2.$$

The theorem is readily derived from this result. □

Yeung, Tang and Vuik also show that the deflated residual can be bounded by the spectrum, if all eigenvalues of A are contained in an ellipse away from the origin.

$$\frac{\|\hat{r}_m\|}{\|\hat{r}_0\|} \leq \min_{\substack{p\in\mathcal{P}_m \\ p(0)=1}} \max_{k>d} \kappa_2(R_{22})|p(\lambda_k)| \leq \kappa_2(R_{22}) \left(\frac{a + \sqrt{a^2 - d^2}}{c + \sqrt{c^2 - d^2}} \right)^m$$

The result is similar to Propositions 2-4.3(b) and 2-4.4. In this case, however, the min-max problem is restricted to the set of eigenvalues $\lambda_k \in \{\lambda_{d+1}, \ldots, \lambda_n\}$, since the first d eigenvalues are zero. Furthermore, the bound depends on $R_2 2$, given in (3-3.13), instead of the eigenvectors V.

In conclusion, as long as Z contains a basis for an A-invariant subspace, deflated GMRES will converge faster than regular GMRES, and the residual is bounded by the undeflated part of the spectrum. Although theoretical proof does not exist, we expect similar behavior when near-invariant subspaces are used. Since the eigenvectors are by construction an invariant subspace of A, the next section introduces methods to approximate the eigenspace.

3-4 Computing the deflation vectors

A number of approaches to compute the deflation subspace Z a proposed in deflation-related literature, with varying degrees of effectiveness depending on the application. We will review each method for its applicability in IX in the upcoming subsections. Before doing so, however, we discuss two aspects of computing the deflation vectors that apply in general.

Firstly, since our linear system is non-symmetric, approximate eigenvalues and eigenvectors are either real or come in complex-conjugate pairs [5, Theorem 1.3]. To retain real arithmetic, a complex-conjugate pair of eigenvectors should be replaced by one eigenvector containing the real part of the complex pair, and one eigenvector containing the imaginary part [20]. For a complex conjugate pair (u_k, u_{k+1}), this is done by using the transformation

$$\begin{bmatrix} u'_k \\ u'_{k+1} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ i & -i \end{bmatrix} \begin{bmatrix} u_k \\ u_{k+1}. \end{bmatrix}$$

The vectors (u'_k, u'_{k+1}) are Schur vectors of A.

Secondly, some cases will allow for the reuse of information from previous deflation vector computations. If a system is propagated over time, it is possible that approximate eigenvectors from previous non-linear iterations can be used again. This approach is only feasible if the system matrix A at a certain non-linear iteration is similar to the system matrix at the previous non-linear iteration [20].

3-4-1 Exact eigenvectors

As each eigenpair (θ, y) of A satisfies

$$Ay = \theta y, \tag{3-4.1}$$

the subspace Z containing d exact eigenvectors of A is by definition an A-invariant subspace. Having met the conditions of Theorem 3-3.1, we can expect equal or faster convergence of GMRES. The obvious drawback of this approach is the cost of computing exact eigenvectors, especially when A is large. In addition, the computational burden grows when the number of extreme eigenvalues increases. Given the grid sizes in reservoir simulation, computing exact eigenvectors is not feasible. As we will discuss next, though, eigenvectors can also be approximated during the course of the inner GMRES iterations. Hence, we are still interested in the effect of exact eigenvector deflation for analytic purposes, as it will provide us with a "best-case" scenario.

3-4-2 Ritz vectors

For an approximate eigenvector z with corresponding eigenvalue θ, the Galerkin orthogonal projection problem [9, 16] states

$$Az - \theta z \perp \mathcal{K}_m.$$

For a basis V_m for $\mathcal{K}_m$, and with $z = V_m y$, this becomes

$$V_m^T (A - \theta I) V_m y = 0. \tag{3-4.2}$$

Using Lemma 2-3.1(b) and the fact that $V_m^T V_m = I$, (3-4.2) reduces to

$$H_m y = \theta y,\ z = V_m y.$$

Ritz vectors approximate the eigenvectors of A. Moreover, the Ritz values tend to approximate the eigenvalues of A. Therefore, we can take the d approximated eigenvectors z corresponding to the d smallest Ritz values as the columns of Z. In terms of eigenvector approximations for extreme eigenvalues, Chapman, Saad [16] and Morgan [17] report that Ritz vectors are outperformed by harmonic Ritz vectors. The latter concept will be introduced next.

3-4-3 Harmonic Ritz vectors

Whereas Ritz vectors are formed by imposing a Galerkin projection, harmonic Ritz vectors are obtained by using the Petrov-Galerkin orthogonality conditions. The approximation error in the eigenvalue problem (3-4.1) for the approximate eigenpair (θ, z) is set orthogonal to the subspace $A\mathcal{K}_m$, i.e.

$$Az - \theta z \perp A\mathcal{K}_m \qquad \Leftrightarrow \qquad (AV_m)^T(Az - \theta z) = 0,\ z = V_m y. \tag{3-4.3}$$

In the following lemma, we show that there are two ways to solve (3-4.3).

Lemma 3-4.1. *Denote $\lfloor B \rfloor$ as the matrix B without its last row. Equation* (3-4.3) *is equivalent to solving:*

(a) the eigenvalue problem $(H_m + h_{m+1,m}^2 H_m^{-T} e_m e_m^T)y = \theta y$;

(b) the generalized eigenvalue problem $R_m y = \theta \lfloor Q_m V_{m+1}^T V_m \rfloor y$.

Proof. First, note that

$$(AV_m)^T AV_m = (V_{m+1}\bar{H}_m)^T V_{m+1}\bar{H}_m = \bar{H}_m^T V_{m+1}^T V_{m+1}\bar{H}_m = \bar{H}_m^T \bar{H}_m,$$

by Lemma 2-3.1(b).In addition,

$$(AV_m)^T V_m = (V_{m+1}\bar{H}_m)^T V_m = \bar{H}_m^T V_{m+1}^T V_m.$$

Combining the latter two identities, we can rewrite (3-4.3) as

$$\begin{aligned}(AV_m)^T(Az - \theta z) = 0 \ &\Leftrightarrow\ (AV_m)^T AV_m y = \theta (AV_m)^T V_m y \\ &\Leftrightarrow\ \bar{H}_m^T \bar{H}_m y = \theta \bar{H}_m^T V_{m+1}^T V_m y. \end{aligned} \tag{3-4.4}$$

It is easily checked that $\bar{H}_m^T V_{m+1}^T V_m = H_m^T$. Consequently, (3-4.4) is equivalent to

$$H_m^{-T} \bar{H}_m^T \bar{H}_m y = \theta y \tag{3-4.5}$$

Furthermore, we have

$$\bar{H}_m^T \bar{H}_m = H_m^T H_m + h_{m+1,m}^2 e_m e_m^T.$$

Hence,

$$H_m^{-T} \bar{H}_m^T \bar{H}_m = H_m^{-T}(H_m^T H_m + h_{m+1,m}^2 e_m e_m^T) = H_m + h_{m+1,m}^2 H_m^{-T} e_m e_m^T,$$

which, combined with (3-4.5), completes the proof of part (a).

To prove part (b), recall the QR decomposition for $\bar{H}_m$:

$$Q_m \bar{H}_m = \bar{R}_m, \text{ with } \bar{R}_m = \begin{bmatrix} R_m \\ 0 \end{bmatrix}.$$

Substituting the result in (3-4.4), we get

$$\bar{H}_m^T \bar{H}_m = (Q_m^T \bar{R}_m)^T (Q_m^T \bar{R}_m) = \bar{R}_m^T Q_m Q_m^T \bar{R}_m = \bar{R}_m^T \bar{R}_m = R_m^T R_m,$$

and

$$\bar{H}_m^T V_{m+1}^T V_m = \bar{R}_m^T Q_m V_{m+1}^T V_m = R_m^T \lfloor Q_m V_{m+1}^T V_m \rfloor.$$

Hence, (3-4.4) is equivalent to

$$R_m^T R_m y = \theta R_m^T \lfloor Q_m V_{m+1}^T V_m \rfloor y,$$

which has the same eigenpairs as

$$R_m y = \theta \lfloor Q_m V_{m+1}^T V_m \rfloor y.$$

□

Although both formulas result in a valid spectrum, approach (b) is preferable for computational reasons. Approach (a) would require keeping an original copy of the matrix H_m in memory, without the Givens rotations. Approach (b), on the other hand, does not have this requirement, and allows the same Givens rotations Q_m that are saved and applied to H_m, to be used on $V_{m+1}^T V_m$.

Note that as the Ritz values converge to the eigenvalues (e.g. Figure 3-1.2), the harmonic Ritz eigenvector approximations approach the true eigenvectors. Therefore, the cycle size m needs to be chosen sufficiently large in order to obtain reasonable approximations. In the results, we vary the value of m in deflated GMRES using harmonic Ritz vectors, and demonstrate the impact on the convergence. Although Z is not sparse in this case, the harmonic Ritz vectors can be computed at a relatively small cost. Since the user only has to specify how many vectors should be included in the deflation operator, the method has a black-box nature.

3-4-4 Domain-based vectors

Subdomain deflation has been introduced by Nicolaides [24] and Mansfield [59, 60]. Let Ω be the computational domain, which is divided into d nonoverlapping subdomains Ω_j, $j = 1, \ldots, d$. After discretization, denoted by subscript h, let x_i be a grid point in the discretized domain Ω_{h_j}. We define the deflation vector z_j corresponding to Ω_{h_j} as

$$(z_j)_i = \begin{cases} 1, & x_i \in \Omega_{h_j} \\ 0, & x_i \in \Omega_h \setminus \bar{\Omega}_{h_j}. \end{cases} \tag{3-4.6}$$

The deflation subspace is defined as $Z = [z_1, \ldots, z_d]$. The vectors in Z are piecewise-constant, disjoint and orthogonal. For this choice of the deflation subspace, the deflation projectors P_1 and P_2 essentially agglomerate each subdomain in a single cell. Hence, subdomain deflation is closely related to domain-decomposition methods and multigrid [8]. For problems in bubbly flow, which have many similarities with the problems in this thesis, the span of the deflation vectors (3-4.6) approximates the span of the eigenvectors corresponding to the smallest eigenvalues [28].

In [61], a time-dependent diffusion equation is investigated for a layered medium representing the earth's crust. Three approaches are used to construct the domain-based deflation vectors. First, the authors require (z_j) to satisfy the finite element discretization of the governing equation on all subdomains with low permeability. The deflation vectors satisfy (3-4.6) for the remaining highly permeable subdomains. This approach is robust for all test problems, yet costly due to the extra solves required. Second, the authors use the vectors (3-4.6) only on the high-permeability layers, and last, (3-4.6) is used for both the high- and low-permeability layers. The latter method turns out to be the most efficient and robust.

Vermolen, Vuik and Segal, use subdomain deflation in [62] to solve a Poisson problem with discontinuous coefficients. In particular, the amount of overlap between subdomains is investigated. For large contrasts in the coefficients, the authors conclude that no overlap is the best choice. For no contrasts, on the other hand, average overlap is superior. This observation gives rise to the so called 'weighted overlap' method, which mimics average and no overlap in the case of no contrasts and large contrasts, respectively. It is shown that the overlap is crucial in approximating the eigenvectors corresponding to the extreme eigenvalues.

If the discontinuities in the computational domain exhibit a complex geometry, subdomain-levelset deflation can be used to guarantee a good approximation of the eigenvectors corresponding to extreme eigenvalues [28]. Whereas subdomain deflation does not take jumps into account, subdomain-levelset deflation identifies different regions in the domain with similar properties. A simple example is given in Figure 3-4.1.

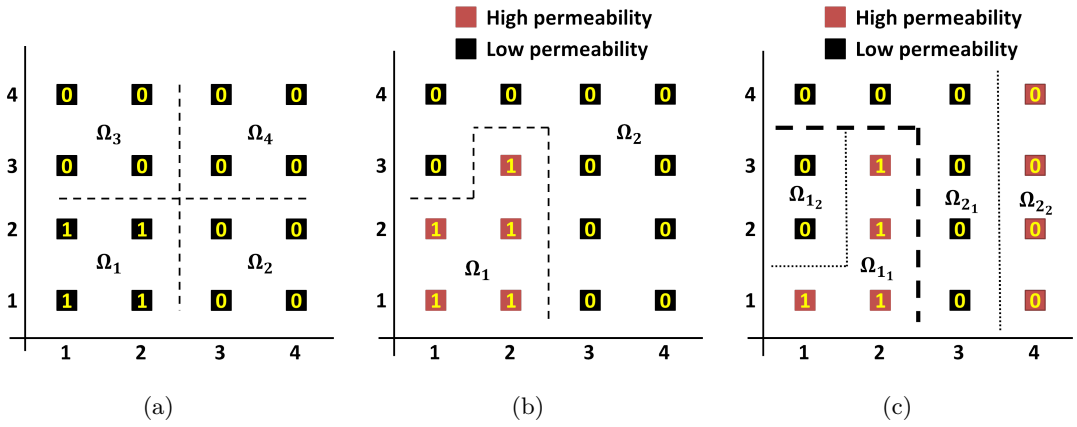


Figure 3-4.1: Subdomain (a), levelset (b) and subdomain-levelset deflation (c).

The grid is 4×4 and nodes are shown as squares. In each case, the values shown on the nodes correspond to the values in the first deflation vectors. In the middle and right figure, the border between the red nodes (high permeability) and black nodes (low permeability) exemplifies a sharp contrast in the PDE coefficient. The figures shows the following:

- In the left figure, subdomain deflation is used. The dashed line divides the domain into the four subdomains $\Omega_1, \Omega_2, \Omega_3$ and Ω_4. Each subdomain corresponds to a unique deflation vector.
- In the middle figure, levelset deflation is used. This time, the dashed line coincides with the contrast in the PDE coefficient. As a result, we get the two domains Ω_1 and Ω_2.
- In the right figure, subdomain-levelset deflation is used. The subdomain division is determined using certain criteria, which in this example leads to the division (dashed line) between Ω_1 and Ω_2. Within each subdomain, levelset deflation (dotted line) uses the jump between the high permeability and low permeability nodes to obtain the subdomains $\Omega_{1_1}, \Omega_{1_2}, \Omega_{2_1}$ and Ω_{2_2}

In Figure 3-4.1, the first deflation vector is shown for each method. If we assume an ordering from left to right and from the bottom to the top, the respective deflation matrices shown next.

$$Z = \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \Omega_1 & \Omega_2 & \Omega_3 & \Omega_4 \end{matrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad Z = \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \Omega_1 & \Omega_2 \end{matrix} \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \quad Z = \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \Omega_{1_1} & \Omega_{1_2} & \Omega_{2_1} & \Omega_{2_2} \end{matrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

In IX, the subdomain devision is determined by the parallel partitioning. Each parallel subdomain corresponds to a processor. In the implementation of our subdomain-levelset deflation algorith, we will always assume that the parallel partitioning is given.

In conclusion, (3-4.6) will be used for both the high- and low-permeability layers. The use of overlap in our deflation vectors is part of ongoing research. Subdomain-levelset deflation is our algorithm of choice, in addition to harmonic Ritz deflation. To apply the levelset method, that is, to identify jumps in the coefficients, a partitioner algorithm has been developed. The pseudocode is discussed in Section 4-2-1.

3-4-5 Solution deflation

In a history-based deflation method, Clemens et al. [19] propose to reuse an optimal linear combination of solutions of the linear system at previous time steps as the deflation operator. As a result, eigenvector components of the exact solution are optimally resolved and the authors achieve significantly improved converge of the CG method for electromagnetic discrete field formulations. On the downside, the method is only effective when the current solution is approximated by the span of solutions at previous timesteps. Due to complex geometries and flow patterns, this is often not the case in reservoir simulation. Z will also be very dense, rendering this approach infeasible for an effective sparse parallel implementation. Hence, solution deflation will not be used.

Chapter 4

Implementation

In the previous section, we identify a number of methods to construct and apply the deflation operator. We conclude that deflation using a projection preconditioner is the most feasible approach, combined with either harmonic Ritz or domain-based deflation vectors. In most of our numerical experiments, deflation is applied to the pressure system, although we have also tried using deflation in the full (reservoir) system matrix.

In this section, the implementation of these two methods will be discussed. The last section of the chapter is devoted to the parallel implementation.

4-1 Harmonic Ritz deflation

The pseudocode for deflated GMRES using harmonic Ritz deflation vectors is given in Algorithm 5.

Algorithm 5 right-preconditioned GMRES - harmonic Ritz deflation

1: Setup $P_1 = P_2 = I$, $x^* = 0$ and flag = false
2: Compute $r_0 = P_1(b - Ax_0)$, $\beta = \|r_0\|_2$, and $v_1 = r_0/\beta$.
3: **for** $j = 1, 2, \ldots, m$ **do**
4: $w_j = P_1 A M^{-1} v_j$
5: **for** $i = 1, \ldots, j$ **do**
6: $h_{i,j} = (w_j, v_i)$
7: $w_j = w_j - h_{ij} v_i$
8: **end for**
9: $h_{j+1,j} = \|w_j\|_2$
10: **if** $h_{j+1,j} = 0$ **or** converged **then**
11: set $m = j$ and **go to** 24
12: **end if**
13: $v_{j+1} = w_j / h_{j+1,j}$
14: **end for**
15: **if** flag = false **then**
16: solve $(AV_m)^T(Ay_k - \theta_k y_k) = 0$ for y_k
17: $z_k = V_m y_k$, for $k = 1, \ldots, z_d$
18: Fill $Z = [z_1 \ldots z_d]$
19: $A_p = AM^{-1}$, $E = Z^T A_p Z$
20: $P_1 = I - A_p Z E^{-1} Z^T$, $P_2 = I - Z E^{-1} Z^T A_p$
21: $x^* = Z E^{-1} Z^T b$
22: Set flag = true
23: **end if**
24: Fill $\bar{H}_m = \{h_{ij}\}$ for $1 \leq i \leq m+1$, $1 \leq j \leq m$.
25: Compute the minimizer u_m of $\|\beta e_1 - \bar{H}_m u\|_2$ and set $x_m = x_0 + M^{-1} V_m u_m$.
26: **if** converged **then** $x_m = P_2 x_m + x^*$ and return **else** set $x_0 = x_m$ and **go to** 2

Algorithm 5 assumes that the deflation vectors are not available at the start of the iteration. Instead, the information generated by GMRES is used to compute the harmonic Ritz vectors. Z is constructed by taking the first d eigenvector approximations y_k in line 16. The generalized eigenvalue problem is solved using Lemma 3-4.1. In the implementation we include a flag that freezes the deflation operator after the first restart. In theory, the harmonic Ritz vectors could be recomputed after every cycle, and appended to Z. This will further improve convergence, as more eigenvalues are deflated from the spectrum, but comes at a significant cost if m (cycle size) is relatively small.

Remark 4-1.1. *A maximum of m harmonic Ritz eigenpairs can be computed from the generalized eigenvalue problem. Hence, we have the constraint $0 \leq d \leq m$.*

Because a full cycle of GMRES is necessary to compute the harmonic Ritz vectors, a restart is needed before we can deflate the spectrum of A. It will become clear in the

results section, that this renders the use of harmonic Ritz vectors infeasible for our purposes.

4-2 Subdomain-levelset deflation

The pseudocode of the main program for deflated GMRES using the subdomain-levelset method is given in Algorithm 6.

Algorithm 6 right-preconditioned GMRES - Static deflation

1: Call Algorithm 7 to compute Z
2: Setup $A_p = AM^{-1}$, $E = Z^T A_p Z$, and $x^* = ZE^{-1}Z^T b$.
3: Setup $P_1 = I - A_p ZE^{-1}Z^T$ and $P_2 = I - ZE^{-1}Z^T A_p$.
4: Compute $r_0 = P_1(b - Ax_0)$, $\beta = \|r_0\|_2$, and $v_1 = r_0/\beta$.
5: **for** $j = 1, 2, \ldots, m$ **do**
6: $w_j = P_1 AM^{-1} v_j$
7: **for** $i = 1, \ldots, j$ **do**
8: $h_{i,j} = (w_j, v_i)$
9: $w_j = w_j - h_{ij} v_i$
10: **end for**
11: $h_{j+1,j} = \|w_j\|_2$
12: **if** $h_{j+1,j} = 0$ **or** converged **then**
13: set $m = j$ and **go to** 17
14: **end if**
15: $v_{j+1} = w_j / h_{j+1,j}$
16: **end for**
17: Fill $\bar{H}_m = \{h_{ij}\}$ for $1 \leq i \leq m+1$, $1 \leq j \leq m$.
18: Compute the minimizer u_m of $\|\beta e_1 - \bar{H}_m u\|_2$ and set $x_m = x_0 + M^{-1} V_m u_m$.
19: **if** converged **then** $x_m = P_2 x_m + x^*$ and return **else** set $x_0 = x_m$ and **go to** 4

The first line in Algorithm 6 calls a function to compute Z. As discussed in 3-4-4, Z can be reused from previous iterations, constructed manually, or computed using our partitioner algorithm. The latter will be discussed in the next section.

The computations in line 2 and 3 involve setting up the deflation operators, and can be executed offline since Z is available a priori. The algorithm clearly illustrates the action of P_1 and P_2. The search space is narrowed to the deflated subspace by solving the deflated system (3-3.10) using right preconditioned GMRES (Algorithm 4). To return to the original space, x_m is multiplied with P_2, and the result is added to x^* (cf. Equation (3-3.12)).

Both in Algorithm 5 and Algorithm 6, the expression $A_p = AM^{-1}$ is used in constructing P_1 and P_2. The multiplication of the system matrix and the inverse of the precondi-

tioner is an expensive matrix-matrix product that is expensive, or even impossible if the preconditioner is not explicitly formed (AMG). Therefore, we investigate taking $A_p = A$ instead. In the results, we show that this is indeed possible. The convergence convergence slightly deteriorates, but deflated GMRES remains faster than non-deflated GMRES.

4-2-1 Partitioner

The goal of the the partitioner is to identify distinct regions of similar permeability values in the computational domain. The implementation was inspired by the work of Lingen et. al [34]. We will use several examples to illustrate the partitioning, and discuss the pseudocode.

Remark 4-2.1. *In the upcoming discussion, we will refer to 'regions' as subdomains in the computational domain. A region consists of one or more cells in a certain range of permeability values. Two regions are 'merged' when the cells from both regions are taken together in a new, combined region. Each region can be used to construct one deflation vector, by assigning ones to the cells inside the region and zeros elsewhere (similar to Figure 3-4.1). Therefore, the terms region and deflation vector are often used interchangeably.*

A deflation vector is assigned to each region. Several criteria are used to develop the code:

- The input data for the partitioner is an adjacency graph of the cells in the computational domain and a vector of permeability vectors.
- The output of the partitioner is a set of deflation vectors.
- Regions with similar permeability values should be merged together.
- After the partitioner algorithm has been applied, the number of remaining regions should be equal to or less than the prespecified number of deflation vectors.
- The borders of the final set of regions should coincide (as much as possible) with the largest coefficient jumps that cause extreme eigenvalues to appear.

The adjacency graph is store as a sparse matrix, and shows which cells are connected. We observe in our experiments that the set of deflation vectors resulting from a successful subdomain-levelset partitioning using the criteria above approximates the extreme eigenvectors well.

We have seen an elementary example of a heterogeneous domain in Figure 3-4.1(c). Figure 4-2.1 shows another example of a domain with distinct regions of permeability. For simplicity, we assume that permeability differences equal to or larger than 20

cause extreme eigenvalues to appear[1]. Smaller differences do not influence the spectrum. Furthermore, we assume that the permeability in x-, y- and z-direction is equal.

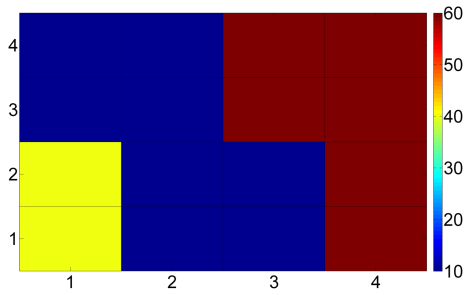


Figure 4-2.1: Example of isolated regions in a heterogeneous domain.

In this 4×4 domain, three regions of connected cells can be identified, with permeability values 10, 40 and 60. Because the regions are separated and their borders coincide with the large permeability jumps, we can easily assign three deflation vectors.

In practice, though, more complex geometries arise. Figures 4-2.2(a) and 4-2.2(b) show two examples.

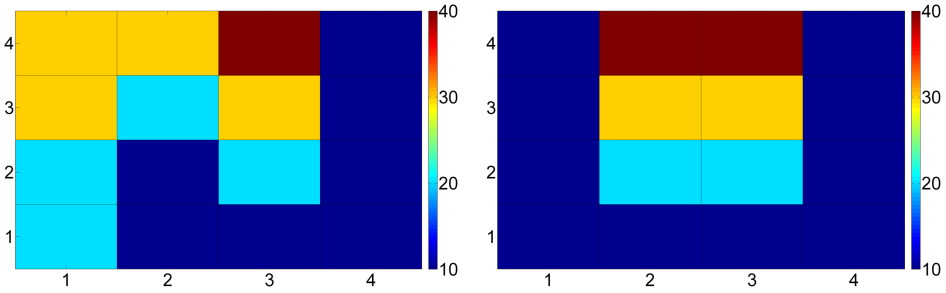


Figure 4-2.2: Examples of heterogeneous domains.

In this case, the regions are not distinct. In the left figure, a large jump exists between the cells with coordinates $(x, y) = (3, 4)$ and $(4, 4)$ because the permeability difference is 30. Moving south from $(3, 4)$ towards $(3, 1)$, however, the difference between each cell is only 10 and the permeability value in $(2, 1)$ is again equal to the value in $(4, 4)$. As a result, cell $(3, 4)$ and $(4, 4)$ are 'connected' by a path of moderately increasing permeability. A similar situation can be observed in Figure 4-2.2(b) in the jumps between $(1, 4)$ and $(2, 4)$, as well as between $(3, 4)$ and $(4, 4)$.

Paths of continuous, moderately increasing permeability appear often in practice and pose a serious difficulty for a partitioner algorithm. Initial attempts to develop a parti-

[1] We will see in Chapter 5 that for real cases this threshold lies around 10^4

tioner relied on a search function that checks, for each cell, if the permeability jump to an adjacent cell is smaller than a certain threshold. If this is the case, then the regions that these cells belong to are merged. As a result, regions that are separated by a jump on one side, but connected by a path of moderately increasing permeability on another side are eventually merged together. Because the heterogeneity is ignored, the resulting deflation vector will not approximate the eigenvector corresponding to the extreme eigenvalue that was caused by the jump.

To mitigate the detrimental effect of moderately increasing permeability paths, we switched from a cell-based check to a region-based check. Instead of comparing permeability values between individual cells, our new algorithm sums the permeability differences of all bordering cells between two regions. The total is then compared to a threshold in deciding whether or not to merge. The algorithm works as follows:

1. Compute the maximum permeability over all cells in the domain
2. Use the maximum to divide the permeability field into a prespecified number of initial regions with a similar range of permeability values
3. Assign a region number to each separated region
4. Sum permeability differences between each pair of adjacent regions
5. Loop over all adjacent region-pairs and merge if the summed jump is smaller than a certain threshold
6. Go to step 4, unless no regions were merged in step 5
7. Increase the threshold stepwise and go to step 4, unless the number of remaining regions is smaller than or equal to the number of desired deflation vectors

In the loop in step 5, it is crucial that we eliminate any regions that have been merged from the iteration. For example, after region i has been compared with region j and the regions are merged, then region j will not be merged again with other regions until we reach step 6. After all adjacent regions of region i have been checked, this region will be removed from the loop as well. This procedure further minimizes the risk of merging distinct regions along a path of moderately increasing permeability.

We will illustrate the partitioning process with the example from Figure 4-2.2(a). Step 1 gives us the maximum permeability of 40. Assuming that we set the initial number of regions to 4, we assign a region number O_k to each permeability value p_i, $i = 1, \ldots, n_c$, using the criteria

$$k\frac{40}{4} < p_i \leq (k+1)\frac{40}{4}, \qquad k = 1, 2, \ldots, 4$$

The result is shown in shown in Figure 4-2.3

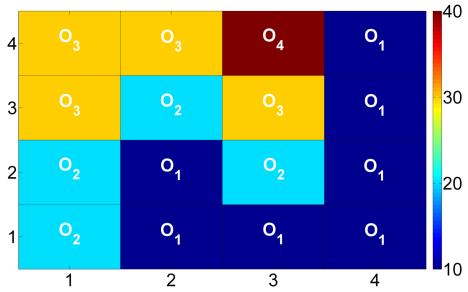


Figure 4-2.3: Step 2 of the partitioner applied to Figure 4-2(a).

Observe that separated regions may be assigned the same number, even though they are not connected. This is undesirable so in step 3 we check if regions in the same permeability range are connected. If not, a new number is assigned. The result is shown in Figure 4-2.4.

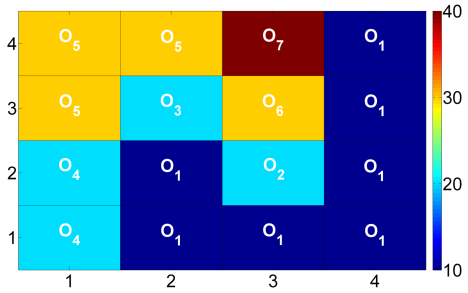


Figure 4-2.4: Step 3 of the partitioner applied to Figure 4-2.3.

In step 4, we iterate over each region and sum the permeability jumps with adjacent regions. Region O_1 in Figure 4-2.4, for example, has a total difference of 30 with region O_2, 10 with region O_3 and 20 with region O_4, 20 with region O_6 and 30 with region O_7. The result of step 4 is saved in an $n \times n$ symmetric matrix W, which we refer to as the 'jump matrix'. In this example, W is the 7×7 matrix

$$W = \begin{bmatrix} 0 & 30 & 10 & 20 & 0 & 20 & 30 \\ 30 & 0 & 0 & 0 & 0 & 10 & 0 \\ 10 & 0 & 0 & 0 & 20 & 10 & 0 \\ 20 & 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 20 & 10 & 0 & 0 & 10 \\ 20 & 10 & 10 & 0 & 0 & 0 & 10 \\ 30 & 0 & 0 & 0 & 10 & 10 & 0 \end{bmatrix}$$

The first row in W contains the jumps between region O_1 and all other regions. Assuming the initial threshold is chosen as 10, step 5 will merge all region-pairs with a summed permeability smaller than or equal to 10. As a result,

- Region O_1 will merge with region O_3
- Region O_2 will merge with region O_6
- Region O_4 will merge with region O_5

Note that even though region O_7 has a permeability jump of 10 with region O_5 and O_6, the latter regions have already been merged with other regions and may not be used again until the threshold is raised. After renumbering the regions, we obtain the four regions in Figure 4-2.5.

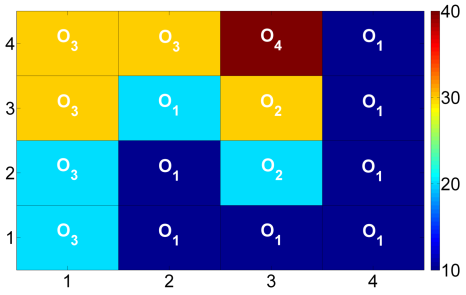


Figure 4-2.5: Step 5 of the partitioner applied to Figure 4-2.4.

The new jump matrix is

$$W = \begin{bmatrix} 0 & 50 & 40 & 30 \\ 50 & 0 & 0 & 10 \\ 40 & 0 & 0 & 10 \\ 30 & 10 & 10 & 0 \end{bmatrix}.$$

Repeating step 5 will cause regions O_2 and O_4 to merge. If the desired number of deflation vectors would be two, then another round of merging regions would add O_3 to the combined region of O_2 and O_4 (in Figure 4-2.5). The final result is shown in Figure 4-2.6.

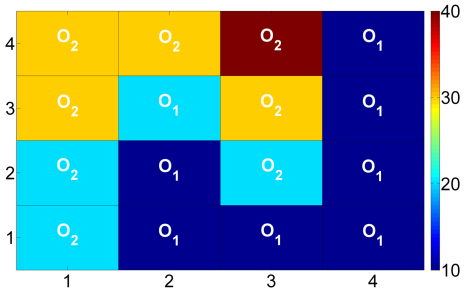


Figure 4-2.6: Result of the partitioning.

It is crucial that the largest differences between permeability values are captured by the deflation vectors. In this case, the large permeability differences between the cells at

$(x, y) = (3, 4)$ and $(4, 4)$, as well as between $(x, y) = (3, 3)$ and $(4, 3)$ are reflected by the two deflation vectors. Consequently, as has been observed in practice, the two deflation vectors in 4-2.6 are good approximations to the eigenvectors corresponding to the jumps. Possible improvements on this algorithm are discussed in section 6-3.

Pseudocode
In [34], Lingen et. al develop a parallel physics-based iterative solver for applications in geomechanical problems. The authors state three objectives:

1. Minimize the overlap between different domains

2. Minimize the variation in the sizes of the subdomains

3. Separate nodes with different material properties

The main difference with our work is the purpose of the algorithm: in [34] the goal is to partition the computational domain in a number of parallel subdomains, whereas in our case the parallel subdomains are given (by IX) and we apply the partitioner as a levelset algorithm (Section 3-4-4). The first two objectives improve parallel performance, as communication, synchronization and workload-distribution are optimized. The third objective is similar to our goal, and "is based on the observation that the preconditioner becomes more effective when the large jumps in material properties coincide with the sub-domain boundaries" [34, p. 9].

The pseudocode is given in Appendix A. Algorithm 8 is nearly identical to [34, Algorithm 1]. As we do not use objective one and two above, and because we are dealing with continuous permeability distributions instead of discrete material property distributions, the other programs in our implementation are different.

The partitioner consists of five functions:

- Main receives the adjacency and permeability input, and computes the maximum.
- Initialize assigns a region number to each cell, based on the permeability range categories.
- Partition merges all adjacent cells with the same number into regions.
- Merge merges adjacent regions until the desired number of regions remains.
- Jumps computes the summed permeability jumps.

Main consecutively calls initialize, partition and merge. Jumps is called by merge to compute the jump matrix W. Figure 4-2.7 contains a schematic overview of the partitioner.

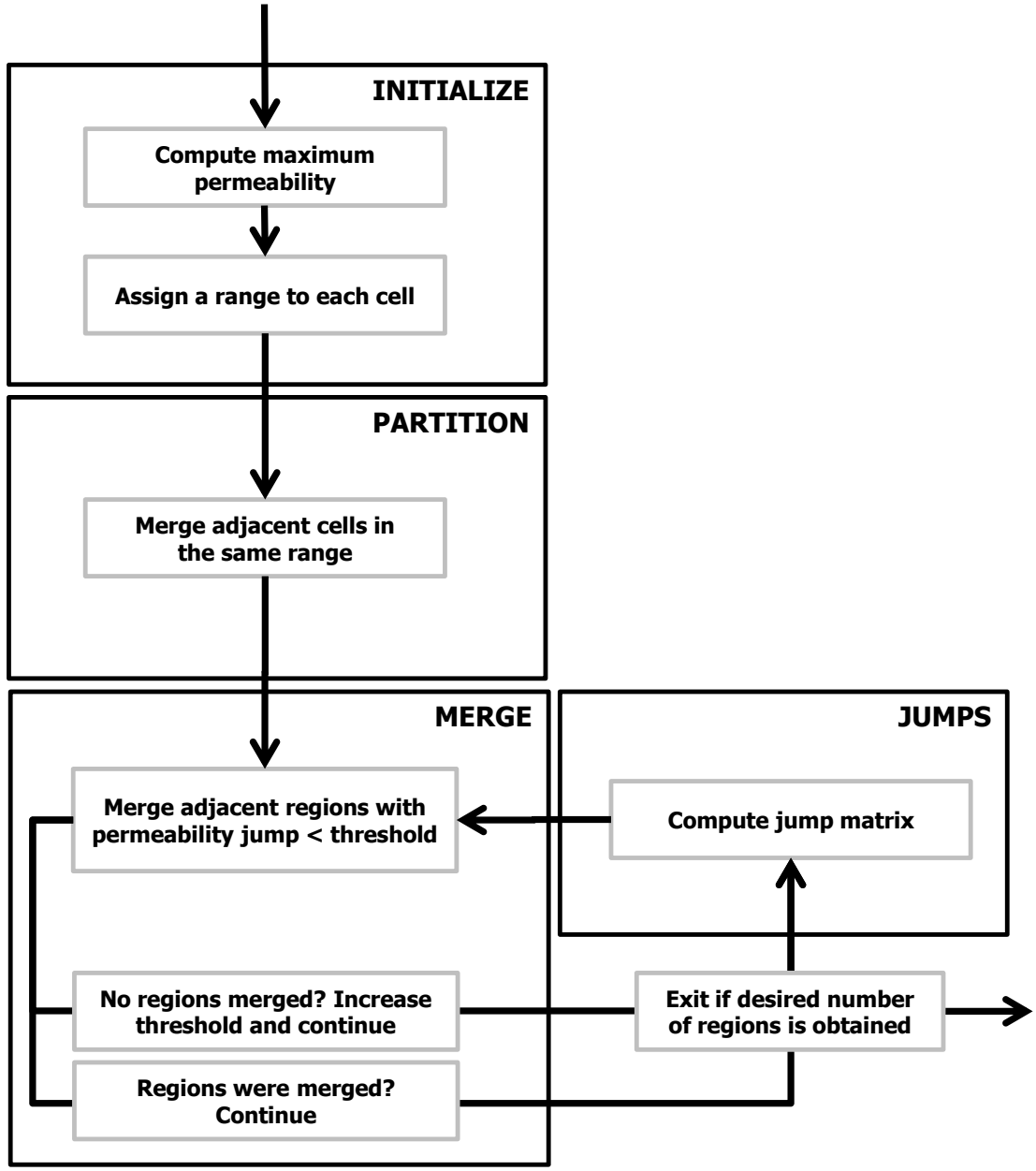


Figure 4-2.7: Overview of the partitioner.

4-3 Parallel implementation

As noted before, the subdomain-levelset deflation method from Figure 3-4.1 is particularly suitable for a parallel implementation. The procedure is as follows:

1. Let the IX engine construct n_p parallel subdomains
2. Compute the maximum permeability over all parallel subdomains
3. Let each processor run the partitioner on the assigned parallel subdomain to obtain n_p sets of deflation vectors $z_1, \ldots, z_{\hat{n}}$, $\hat{n} \leq n_d$.
4. Assign the i'th set of deflation vectors to the i'th processor ($i = 1, \ldots, n_p$).
5. Construct and apply P_1 and P_2 in parallel

The parallel subdomains in IX are constructed using the PARmetis software [81]. In particular, the transmissibilities[2] determine the weights between gridcells. For more information, we refer to [2]. The local deflation vectors (i.e. on a particular parallel subdomain) are appended with zeros on the nodes outside the subdomain to obtain a global set of deflation vectors. Each deflation vector has non-zero values only on the corresponding parallel subdomain, which allows for (cheap) local computations.

We will highlight the parallel computations for the operation $P_1 A v$, for some $v \in \mathbb{R}^{n_c \times 1}$. The computations required for $P_1 v$ and $P_2 v$ can be derived from this procedure. The parallel implementation of the preconditioning is beyond the scope of this thesis.

To align the non-zero blocks in the deflation vectors with the appropriate submatrices in A, we reorder the linear system as

$$Ax = \begin{bmatrix} A_{[1,1]} & \cdots & A_{[1,n_p]} \\ \vdots & \ddots & \vdots \\ A_{[n_p,1]} & \cdots & A_{[n_p,n_p]} \end{bmatrix} \begin{bmatrix} x_{[1]} \\ \vdots \\ x_{[n_p]} \end{bmatrix} = \begin{bmatrix} b_{[1]} \\ \vdots \\ b_{[n_p]} \end{bmatrix} = b.$$

Row i of A is stored on processor i. Each block $A_{[i,j]}$, $1 \leq i, j \leq n_p$ with $i \neq j$, represents the connections between local cells in parallel subdomain i with the 'virtual' cells of subdomain j. The non-zero elements of $A_{[i,j]}$, $i \neq j$, are referred to as 'halo' cells.

The parallel implementation of the deflation operators is inspired by the work of Frank, Vuik and Tang [8, 79]. The authors developed a parallel framework for the deflation vectors defined in (3-4.6). As we do not assume that each node in a parallel subdomain is assigned the value one, our approach is slightly different. Cells in a parallel subdomain can be either one or zero, but, similar to [8, 79], cells outside the subdomain are always zero.

For $P_1 A v$, we have to compute

$$P_1 A v = (I - AZ(Z^T AZ)^{-1} Z^T) A v.$$

The following steps are required:

[2]The transmissibility (or transmissivity) is the ability of a water-bearing unit of a given thickness to transmit a fluid

1. Communicate the non-zero elements of all deflation vectors to each processor.
2. Perform the matrix vector multiplication $y_1 = Av$. On each processor, v is multiplied with the local cells and the halo cells. The result is stored locally.
3. Perform the inner products $y_2 = Z^T y_1$. Because the deflation vectors are zero outside the corresponding parallel subdomain, this inner product can be done locally.
4. Compute $E = Z^T AZ$. An example is shown in Figure 4-3.1 for two processors with two deflation vectors each. As shown in the upper right and bottom left corner of the result, the first processor requires the deflation vectors assigned to the second processor and the second processor requires the deflation vectors assigned to the first processor. This is why we introduced the 'halo deflation vectors' in step one. In practice, only the non-zero elements of the deflation vectors are stored.

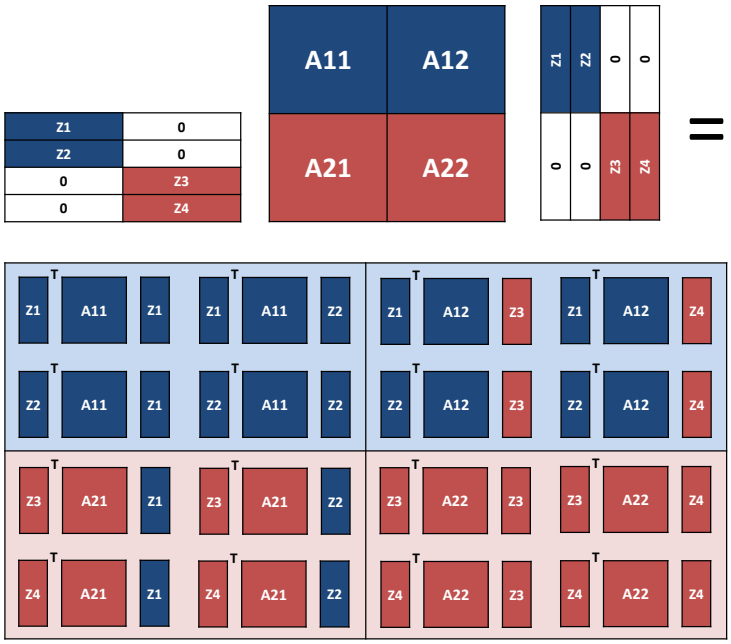


Figure 4-3.1: Parallel computation of the Galerkin matrix.

5. Gather $Ey_3 = y_2$ on one processor and solve using a LU-factorization. The result is distributed.
6. Perform the matrix vector multiplication $y_4 = Zy_3$
7. Perform the matrix vector multiplication $y_5 = Ay_4$. The result is stored locally.
8. Perform the vector update $P_1 Av = y_1 - y_5$.

Compared to [8], step six and seven are slightly modified. Frank and Vuik first compute $w_i = Az_i$ for $i = 1, \ldots, n_p$ and distribute the result across all processors. The multiplication with y_3 can then be done locally. This approach is slightly more efficient, but we found the separate multiplication with Z and A more feasible in IX.

For a fixed computational domain, the parallel subdomains will become smaller as we increase the number of processors. Hence, the partitioner will be faster. Moreover, for a fixed number of deflation vectors per parallel subdomain, the partitioner might be more efficient in capturing the details of the permeability jumps. Whereas the preconditioning generally performs worse for a larger number of processors, we could expect the partitioner to perform better. Two questions arise from this discussion:

- What happens if $n_p \to n_c$, i.e. the number of processors becomes equal to the number of cells in the domain?
- What is the effect of parallel subdomain boundaries crossing through jumps of permeability?

The first question is relatively straightforward to answer. When $n_p = n_c$, we get $Z = I$, and

$$\begin{aligned} P_1 &= I - AZ(Z^T AZ)^{-1} Z^T = 0, \\ P_2 &= I - Z(Z^T AZ)^{-1} Z^T A = 0, \\ x^* &= Z(Z^T AZ)^{-1} Z^T b = A^{-1} b. \end{aligned}$$

Substituting these results into Algorithm 6 gives $x_m = x^* = A^{-1}b$. Hence, we converge in one iteration at the cost of computing the inverse of A (on one processor).

The second question is more difficult to answer. For $n_p = 1$, no jumps are separated but it can be difficult to capture the dominant heterogeneity in a complex and detailed permeability field. For $n_p = n_c$, each individual cell is a deflation vector and the full matrix inverse needs to be computed in the deflated GMRES method. To assess the impact of intermediate choices of n_p, for which we often observe parallel subdomain boundaries crossing through permeability jumps, we will conduct several small Matlab experiments in the results section.

Chapter 5

Results

Harmonic Ritz and subdomain-levelset deflation are implemented both in Matlab and IX. The experiments are run on five cases, varying in size and complexity. Coding in IX is done using C++, and results are visualized using the Schlumberger Eclipse Office viewer.

This chapter is set up as follows. In Section 5-1, we discuss the five cases in terms of the dimensions, initial conditions, wells and performance in the Newton-Raphson method. We continue with an analysis of spectrum in Section 5-2, highlighting the occurrence of isolated and extreme eigenvalues. In Sections 5-3 and 5-4, we discuss the numerical experiments with harmonic Ritz deflation and physics-based deflation, respectively. In the last section, we briefly mention two other deflation strategies.

5-1 Case descriptions

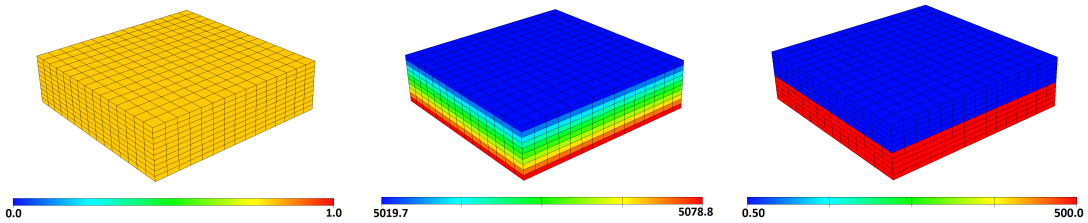
In the upcoming subsections, each case will be discussed in terms of dimensions, wells and, as an indicator for the complexity, the performance of the non-linear solve. The initial conditions for the saturation, pressure and permeability are also highlighted. An overview of the units for the physical variables can be found in Table 1-3.2. For a full overview of all variables and units available in IX, we refer to [2, Appendix A.2]. In addition, we introduce the reporting time T. Note that T often does not coincide with the time steps in the simulation (simulation time), i.e. only a few simulation times are picked as reporting times.

5-1-1 BO

The Black Oil (BO) model is the first case under consideration. A single permeability jump of moderate size exists between the top and bottom layers. Although the BO case is not complex, a moderate amount of time is needed for the solve due to the size of the model.

Dimensions
The size of the BO case is $15 \times 15 \times 10$. All $2,250$ cells hold four unknowns each. Of the four unknowns, one is the pressure variable. Hence, there are $4 \times 2,250 = 9,000$ rows in the full system matrix, and $2,250$ rows in the pressure matrix.

Initial conditions
The initial oil saturation, pressure distribution and permeability (equal in x, y, z directions) are shown in Figure 5-1.1(a), 5-1.1(b) and 5-1.1(c), respectively.



(a) (b) (c)

Figure 5-1.1: Oil saturation, pressure distribution and permeability field of the BO case.

The oil saturation is approximately 0.8. As illustrated in Figure (b), gravity causes an increase in pressure of about 60 Pa towards the bottom layers. In Figure (c), we see that a relatively small permeability jump is present between the fifth (red) and sixth (blue) horizontal layer.

Wells
The model contains nine wells: seven producers and two injectors. The injectors inject water at a fixed rate throughout the simulation. Details on the location of the wells are not relevant, as we analyze this case in Matlab only.

Non-linear iterations
The total time of the simulation is set at 92 days. In figure 5-1.2, the number of cumulative non-linear (Newton) iterations is plotted on the vertical axis.

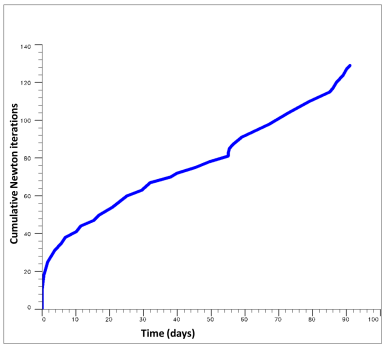


Figure 5-1.2: Cumulative non-linear iterations for the BO case.

For the first five days, the time steps are kept relatively small. Between ten and ninety days, the increase in Newton iterations is roughly linear, which indicates that the problem is weakly non-linear. In general, the Newton-Raphson method can take larger time steps for such problems.

5-1-2 SPE5

The fifth comparative solution project of the Society of Petroleum Engineers (SPE5) is part of a series of comparative solution problems designed to compare reservoir simulators from different companies, research institutes and consultants in the petroleum industry [21]. SPE5 focuses on the simulation of the (miscible) flooding of a reservoir. Details of the particular SPE5 case under consideration are shown below.

Dimensions
The dimensions of the SPE5 case are $7 \times 7 \times 3$. Each cell contains four unknowns, of which one is the pressure. Excluding the well variables, the full matrix has $4 \times 147 = 588$ rows and the pressure matrix has 147 rows.

Initial conditions
The initial permeability in the x, y and z direction is shown in Figure 5-1.3.

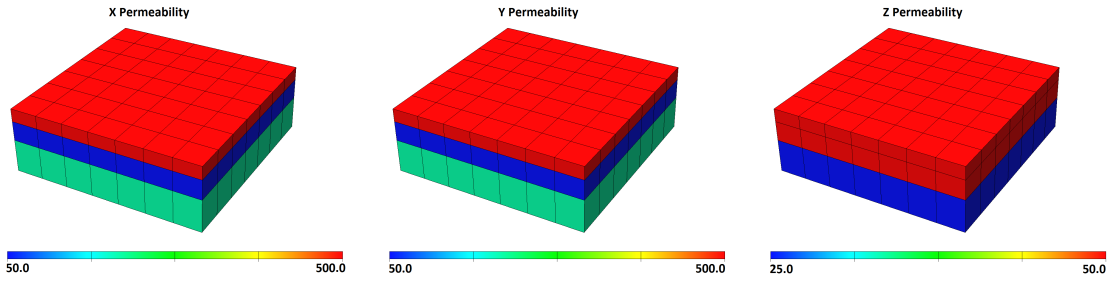


Figure 5-1.3: Permeability in x, y and z-direction.

In vertical direction, the permeability jump between the second and third layer is small. The horizontal layers have moderate heterogeneity, although the permeability is constant in both the x and y direction.

Wells

One injector and one producer are placed in opposite corners of the reservoir. Starting at $T = 0$ days, the injector pumps water in the reservoir, which pushes the oil towards the producer. The oil saturation at time $T = 1, 5$ and 10 is shown in Figure 5-1.4.

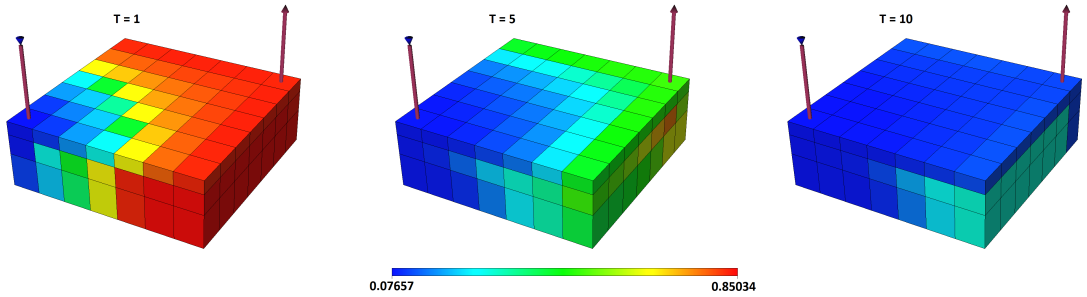


Figure 5-1.4: Oil saturation at $T = 1, T = 5$ and $T = 10$.

We can observe the effect of the initial permeability in x- and y- directions: the middle layer at $T = 5$ has the most oil remaining (lowest permeability in Figure 5-1.3), followed by the bottom layer, in turn followed by the top layer (highest permeability). Some oil remains stuck in the corners of the reservoir at the end of the simulation.

The water saturation is shown in Figure 5-1.5 for $T = 1, T = 5$ and $T = 10$.

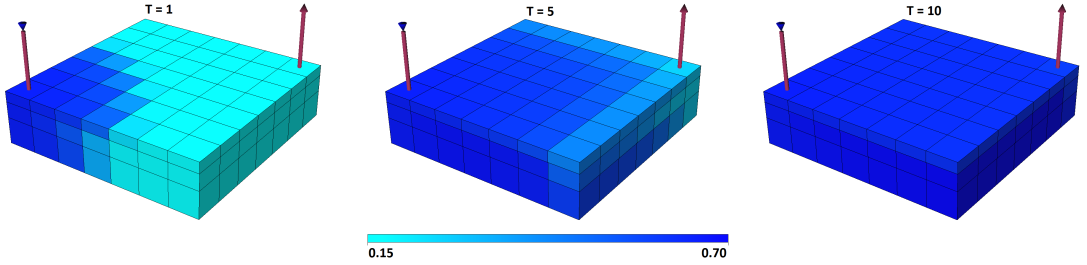


Figure 5-1.5: Water saturation at $T = 1, T = 5$ and $T = 10$.

Comparing the middle images of Figure 5-1.5 and Figure 5-1.4, we see that the injected water has reached most of the reservoir at $T = 5$, even though much of the oil in the right corner has not been extracted yet. At $T = 10$, the reservoir contains roughly 70% water.

Non-linear iterations

The total time of the simulation for the SPE5 case is about ten years. Because the

heterogeneity is relatively weak, and the wells do not introduce any strong non-linearity, the simulation still runs fast. The cumulative Newton iterations plot in Figure 5-1.6 shows that after the slow start-up time (first 30-40 days), the simulation progresses without any problems.

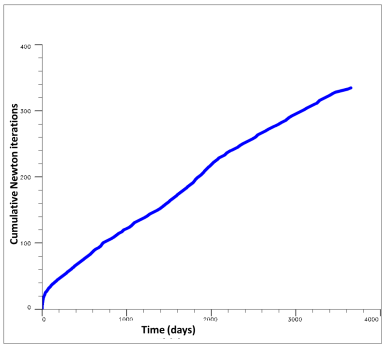


Figure 5-1.6: Cumulative non-linear iterations for the SPE5 case.

5-1-3 SAGD-SMALL

A technique called Steam Assisted Gravity Drainage (SAGD) uses steam injection to create a steam chamber around the producers [22]. The reservoir is heated to make the oil less viscous, after which water is injected into the reservoir. The water evaporates to become steam, creating the steam chamber. After expanding the steam chamber upwards, gravity causes the heavy oil to flow down to the production wells. Due to the steam injection and temperature effects, large pressure gradients occur around the injector and the producer. The gradients, in turn, produce strong non-linearities in the solution of the SAGD case. Hence, the problem is relatively hard to solve.

Dimensions
The SAGD-SMALL case has dimensions $41 \times 1 \times 85$. Similar to the SPE5 case, each cell contains four unknowns. The pressure variable gives rise to a $3,485 \times 3,485$ pressure system matrix. Excluding the well variables, the system matrix has $4 \times 3,485 = 13,940$ rows.

Initial conditions
The initial permeability for the SAGD case is the same in x and y direction. For the z direction, the pattern is the same but the permeability values are halved. Figure 5-1.7 shows the horizontal layer structure that is commonly found in petroleum reservoirs.

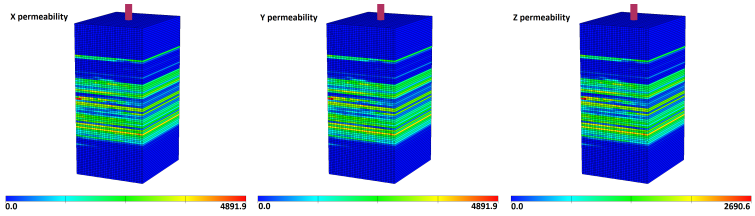


Figure 5-1.7: Permeability in x, y and z directions.

The top and bottom half of the reservoir have zero permeability. Near the center, we see several large permeability jumps of order 10^3.

Wells

Two wells, both capable of acting as a producer and an injector, are placed above each other near the bottom of the reservoir. The wells are visible in the left illustration below. In this figure, we zoomed in to visualize the viscosity of the oil. As time progresses, heating of the reservoir causes the viscosity of the oil around the producers to drop (i.e. oil flows more easily).

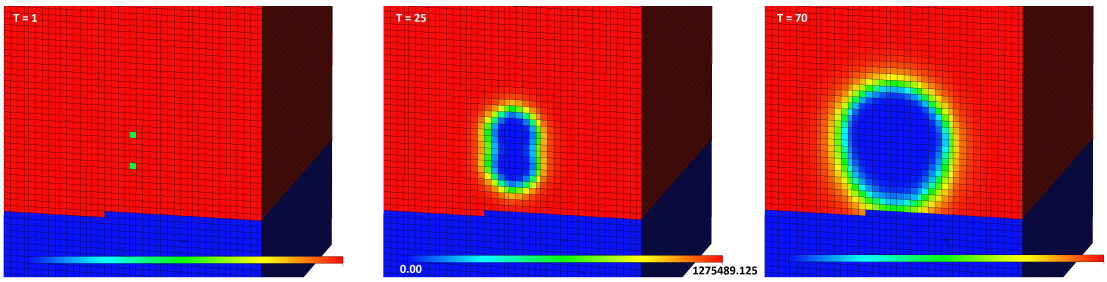


Figure 5-1.8: Oil viscosity at $T = 1, T = 25$ and $T = 70$.

When the producers are switched on, at around T = 10 days, the oil saturation starts to decrease. This is illustrated in Figure 5-1.9.

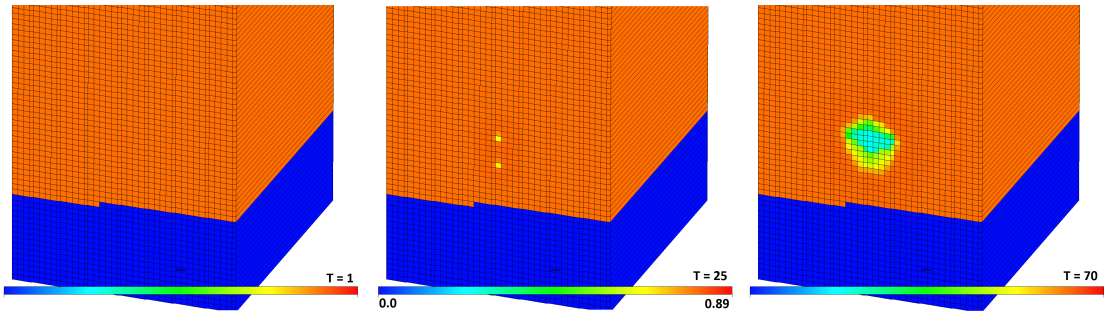


Figure 5-1.9: Oil saturation at $T = 1, T = 25$ and $T = 70$.

Note that the SAGD case is slightly simplified; in real scenarios the oil saturation would not be as constant as shown in Figure 5-1.9. When the oil is produced between $T = 10$ and $T = 70$, the oil saturation around the producer drops from approximately 0.8 to 0.2.

The pressure field at $T = 1, 10, 25, 43, 55, 70$ is shown in the next figure.

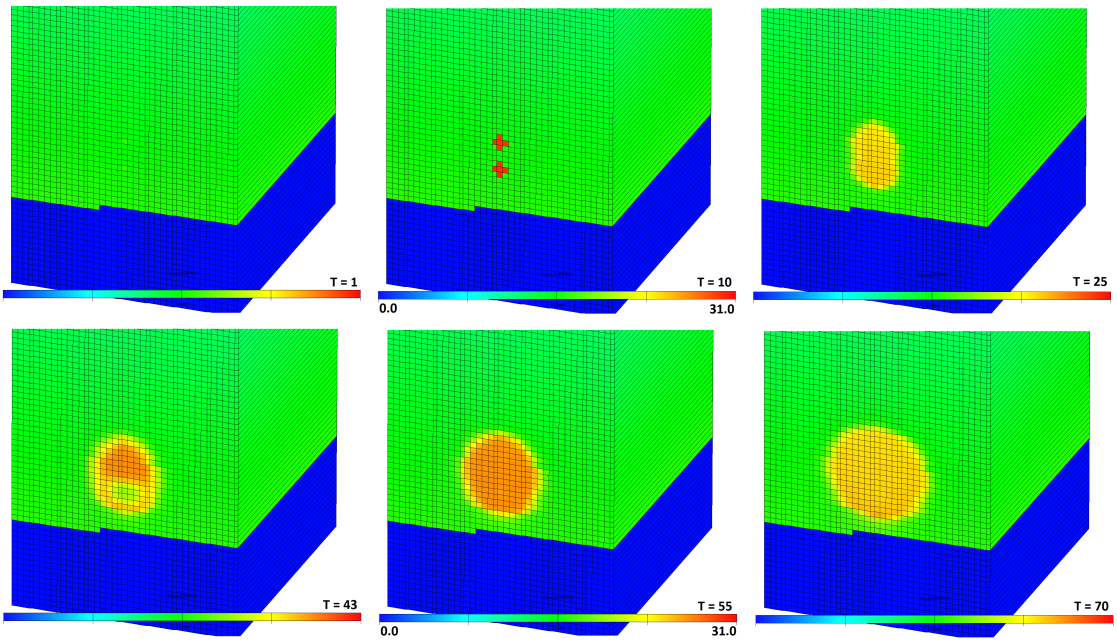


Figure 5-1.10: Pressure at $T = 1, 10, 25, 43, 55, 70$.

The gravity pressure is not visible in this figure, because the pressure changes caused by the steam injection are dominant. After $T = 1$, there is a sudden rise in pressure as the heating is switched on. The pressure from the heating expands as the heating continues, until about $T = 43$, when the bottom well starts producing and the pressure briefly drops. As the steam is injected, the pressure field expands upwards.

Non-linear iterations

The total total time of the simulation is six months. Approximately halfway in, the steam is injected, causing issues for the non-linear solver. As shown by the increased slope in Figure 5-1.11, the amount of Newton iterations increases relatively fast.

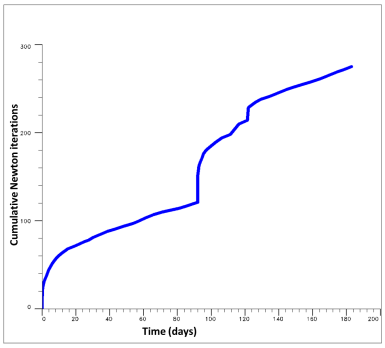


Figure 5-1.11: Cumulative non-linear iterations for the SAGD-SMALL case.

Up to 92 days, the point at which steam is injected, the Newton-Raphson uses time steps of about 5 days. When the steam is injected, the sudden increase in complexity (more high pressure gradients) of the case requires smaller time steps. The sharp increase in Newton iterations shown in Figure 5-1.11 is caused by several failed non-linear solves in which the linear solver did not convergence to a sufficient accuracy. To decrease the complexity and guarantee convergence of the linear solve, the time step is reduced. A similar situation occurs after roughly 125 days.

5-1-4 SAGD-MEDIUM

The SAGD-MEDIUM case is an upscaled version of SAGD-SMALL. The reservoir is expanded in the y direction, to increase the number of layers from 1 to 4. In addition, the wells are drilled horizontally.

Dimensions The SAGD-MEDIUM case has dimensions $41 \times 4 \times 85$. Each cell contains again four variables. The size of the pressure matrix is $13,940 \times 13,940$. Excluding the well variables, the system matrix has $4 \times 13,940 = 55,760$ rows. In this 3-dimensional case, a 7-point stencil is used, which implies that each (block) row of A contains 6 non-zero elements.

Initial conditions
For $y = 0$, the permeability field is identical to the SAGD-SMALL case. As before, the permeability is equal in x and y direction.

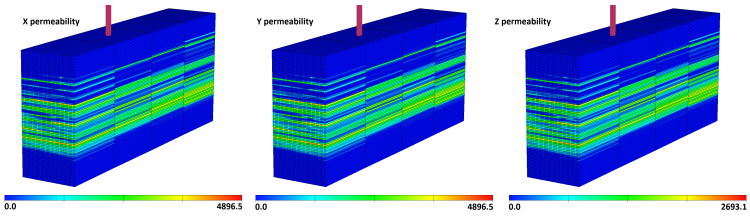


Figure 5-1.12: Permeability in x, y and z directions (SAGD-MEDIUM).

Wells

To illustrate the effect of the horizontal wells, the oil viscosity (left) and oil saturation (right) at the end of the simulation are shown in Figure 5-1.13.

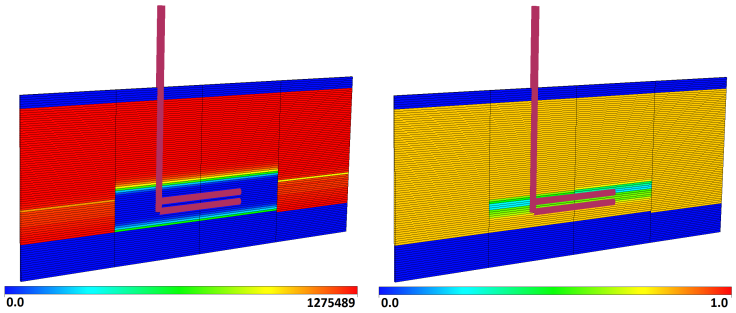


Figure 5-1.13: Oil viscosity (left) and oil saturation (right) at the end of the simulation (SAGD-MEDIUM).

The heating, steam injection and oil production is dominant in the middle two vertical layers.

Non-linear iterations

As shown in Figure 5-1.14, the steam injection causes Newton-Raphson to stall at 92 days. After the time steps are sufficiently reduced, the simulation continues as before. Comparing Figure 5-1.14 to 5-1.11, we see that the sudden cumulative increase of Newton iterations around 92 days is larger in the SAGD-MEDIUM case. This indicates that the increase in size of the domain increases the complexity of the steam injection. In particular, the complexity can be attributed to the fact that the horizontal wells reach a larger part of the reservoir.

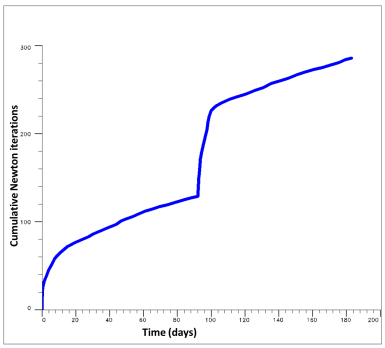


Figure 5-1.14: Cumulative non-linear iterations for the SAGD-MEDIUM case.

5-1-5 SAGD-LARGE

The SAGD-LARGE case is the biggest case used in this thesis. The reservoir is an upscaled version of the SAGD-MEDIUM case.

Dimensions
The SAGD-LARGE case dimensions $125 \times 18 \times 85$. Each cell contains four unknown. The pressure matrix is $191,250 \times 191,250$. Excluding the well variables, the system matrix has $4 \times 191,250 = 765,000$ rows. Similar to the SAGD-MEDIUM case, there are 6 non-zero elements per block-row of A.

Initial conditions
Similar to the SAGD-SMALL and SAGD-MEDIUM cases, the permeability is halved in the z direction. The permeability field is shown in Figure 5-1.15.

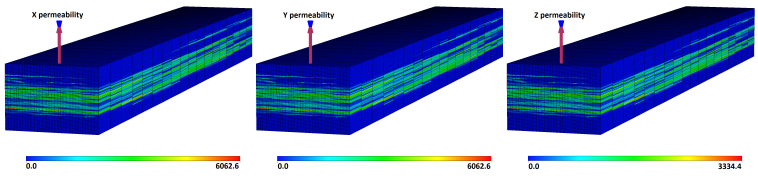


Figure 5-1.15: Permeability in x, y and z directions (SAGD-LARGE).

Wells
The wells are extended in the y direction to cover most of the reservoir. The oil viscosity and oil saturation at the end of the simulation are shown in Figure 5-1.16.

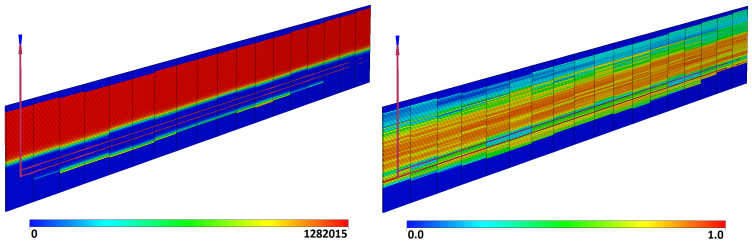


Figure 5-1.16: Oil viscosity (left) and oil saturation (right) at the end of the simulation (SAGD-LARGE).

Non-linear iterations
Figure 5-1.17 shows the cumulative Newton iterations plot for the SAGD-LARGE case.

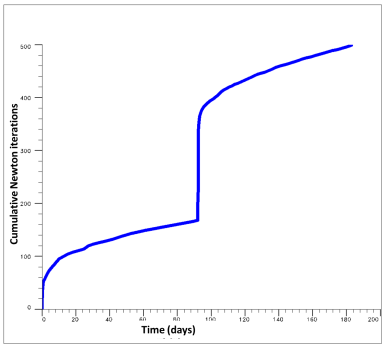


Figure 5-1.17: Cumulative non-linear iterations for the SAGD-LARGE case.

Between 90 and 100 days, the cumulative amount of Newton iterations rises by approximately 240 iterations. The increase is significantly larger than the increase in the SAGD-MEDIUM and SAGD-SMALL cases, where the increase is about 100 and 60 iterations, respectively. Not only is the SAGD-LARGE simulation computationally more demanding due to the size of the reservoir, the convergence issue caused by the steam injection is also magnified due to the relatively large number of cells reached by the wells.

5-2 Eigenvalues and eigenvectors

In the mathematical framework of this thesis, we link the convergence speed of GMRES to the occurrence of extreme eigenvalues. Natural deflation is illustrated with the convergence of the Ritz values in the introduction of Chapter 3, giving rise to the search for artificial deflation algorithms. In this section, we analyze the spectrum of our cases. We compare the spectrum of the BO case to the example at the start of Chapter 3, analyze the eigenvalues and eigenvectors of the SPE5 case and modify the SPE5 permeability field to illustrate the effect of permeability jumps. Furthermore, we investigate the impact of heating and steam injection on the spectrum of the SAGD-SMALL case, and, finally, show the impact of the (parallel) preconditioning on the spectrum of the full matrix.

Most of the eigenvalue and eigenvector plots in this section are generated in Matlab, after exporting the system matrices (at a particular simulation time) from IX. Unless noted otherwise, the spectral information of the diagonally scaled pressure matrix is used. Diagonal scaling clusters the spectrum around one, and allows for straightforward identification of the harmful eigenvalues. The full (reservoir) matrix generally produces complex eigenvalues, due to the lack of symmetry and the mixed characteristics of the different unknowns.

Remark 5-2.1. *We refer to 'isolated' eigenvalues as eigenvalues which are separated from the main cluster. In particular, 'extreme' eigenvalues are separated by at least a factor* $\mathcal{O}(10^2)$.

5-2-1 BO spectrum

The spectrum of the BO pressure matrix at the end of the simulation is shown in Figure 5-2.1.

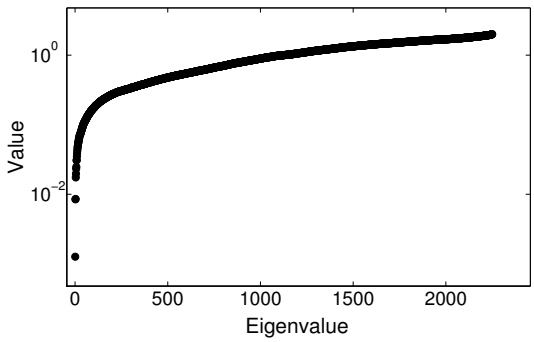


Figure 5-2.1: Eigenvalues of the diagonally scaled BO pressure matrix.

Most of the spectrum is continuous, stretching from $\mathcal{O}(10^{-2})$ to $\mathcal{O}(10^1)$. Two eigenvalues are isolated. Most likely, in line with the findings in [11], the smallest isolated eigenvalue is caused by the single permeability jump shown in Figure 5-1.1(c). Compared to the example at the start of Chapter 3, the gap between the smallest eigenvalue and the rest of the spectrum is smaller than the gap in the spectrum of Figure 3-1.2(c). The effect on the convergence of GMRES, however, is still detrimental, as shown in Figure 5-2.2.

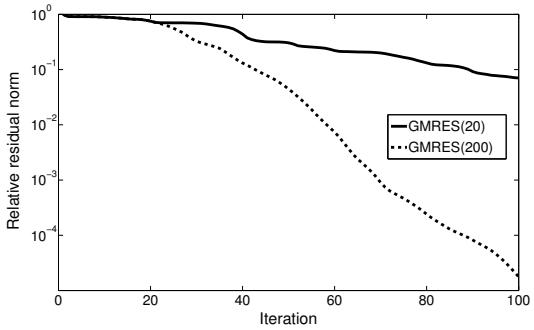
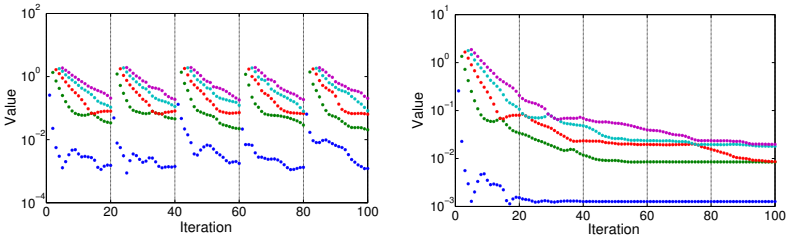


Figure 5-2.2: Convergence of restarted and non-restarted GMRES for the BO pressure matrix.

Unlike the stalling convergence in Figure 3-1.1, the residual of GMRES(20) in this case

is decreasing. Again, when we prevent a restart by setting the cycle size (m) sufficiently high, the convergence improves. The convergence of the Ritz values for GMRES(20) and GMRES(200) is plotted in Figure 5-2.3.



(a) (b)

Figure 5-2.3: Five smallest Ritz values for (a) GMRES(20) and (b) GMRES(200).

Observe that the smallest Ritz value of GMRES(20) nearly converges to the smallest eigenvalue before the restart. As a result, GMRES(20) in Figure 5-2.2 is not stalling. The Ritz values of the example in Figure 3-1.2 do not approach the smallest eigenvalues within 20 iterations, which causes the stalling convergence of GMRES(20) in Figure 3-1.1.

From the comparison we hypothesize that if eigenvalues are more extreme and isolated, it becomes harder for the Ritz values to converge. Because the convergence of the Ritz values affects the convergence of GMRES [7], linear systems with isolated and extreme eigenvalues are more difficult to solve with restarted GMRES. If the eigenvectors corresponding to the extreme eigenvalues can be approximated at a low cost, artificial deflation is a promising method to improve the performance of the linear solver.

5-2-2 SPE5 spectrum

Before we analyze the eigenvalues of the SPE5 pressure matrix, we first discuss the eigenvectors. For each reporting time ($T = 1, \ldots, 10$), the eigenvector corresponding to the smallest eigenvalue is plotted. The result is shown in Figure 5-2.4.

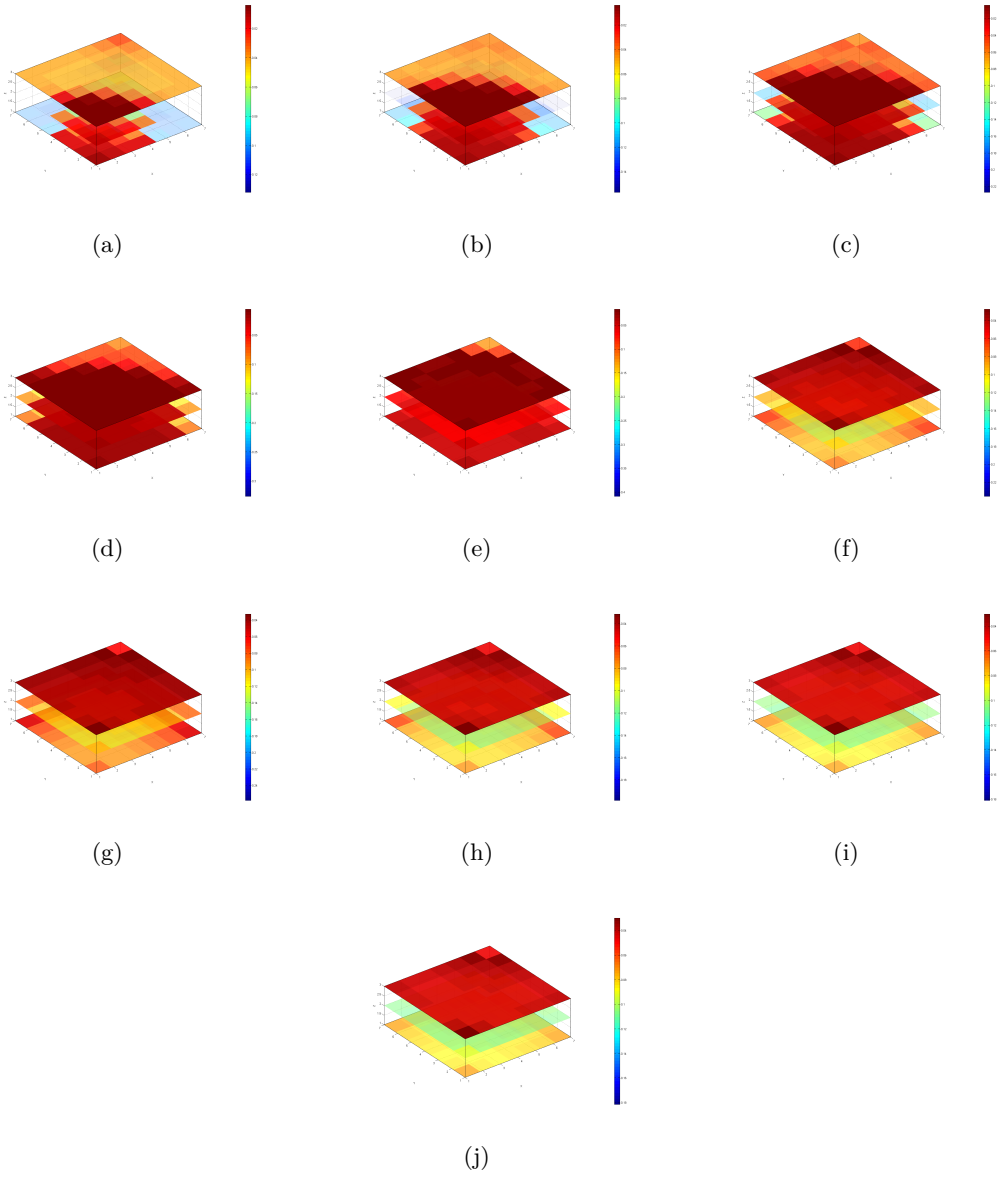
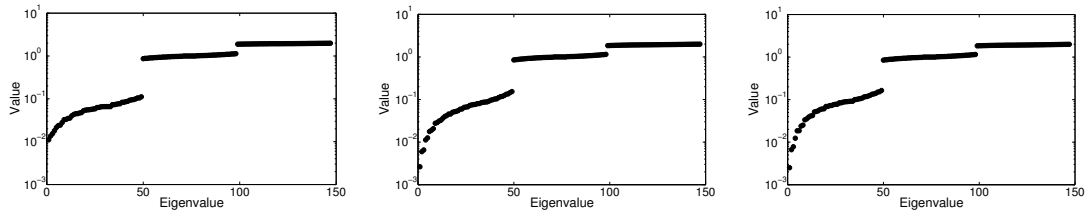


Figure 5-2.4: Eigenvectors of the diagonally scaled SPE5 pressure matrix at $T = 1, \ldots, 10$.

Firstly, observe that the eigenvectors in Figures (a) - (f) contain a flow pattern. Comparing the eigenvectors to the injected water flow in Figure 5-1.5 (rotated 90°), we see that the patterns match. This observation gives rise to a physics-based deflation algorithm that uses the saturation, instead of the permeability, to approximate the eigenvectors corresponding to the extreme eigenvalues. The results are described in Section 5-5-1.

Secondly, observe that the eigenvectors in Figures (g) - (j) reach an equilibrium. After most of the reservoir is saturated with water, three layers become visible in the eigenvectors. In particular, we see a similarity with the x and y permeability in Figure 5-1.3. The top horizontal layer of the eigenvector holds the highest values, the middle layer holds the lowest values and the bottom layer holds the middle values. In each layer, the eigenvector is approximately constant. We conclude that for reporting times $T = 7, \ldots, 10$, three deflation vectors, corresponding to the three horizontal layers, would constitute a suitable approximation of the eigenvector corresponding to the smallest eigenvalue.

As shown in Figure 5-2.5, the (sorted) spectrum of the diagonally scaled SPE5 pressure matrix consists of three eigenvalue clusters.



(a) (b) (c)

Figure 5-2.5: Eigenvalues of the diagonally scaled SPE5 pressure matrix at $T = 1$, $T = 5$ and $T = 10$.

At all three reporting times, the first fifty eigenvalues are separated from the rest of the spectrum. The gap between the other two clusters of fifty eigenvalues is relatively small. As time progresses, eigenvalues in the left tail of the spectrum decrease in value. Comparing these results to the permeability field, we could claim that the large eigenvalue gap corresponds to the large permeability jump (≈ 450) between the top and middle layer, while the small eigenvalue gap corresponds to the small permeability jump (≈ 200) between the middle and bottom layer. This implication, however, contradicts the general consensus in literature (e.g. [11]) that one large coefficient jump causes one extreme eigenvalue to appear in the spectrum (rather than an isolated cluster).

In an attempt to explain this discrepancy, we modify the spectrum of the SPE5 case. A layer of low permeability is sandwiched between two layers of high permeability. The result is shown in Figure 5-2.6.

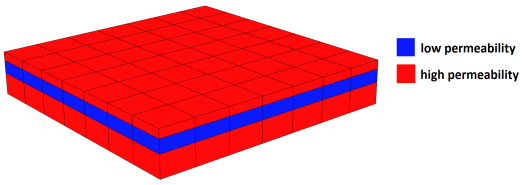


Figure 5-2.6: Modified SPE5 permeability field.

Cells in the middle layer are assigned permeability value 10^0, while we vary the high permeability in the bottom and top layer. The spectra for the permeability jumps 10^1, 10^2, 10^3 and 10^4 are plotted in Figure 5-2.7.

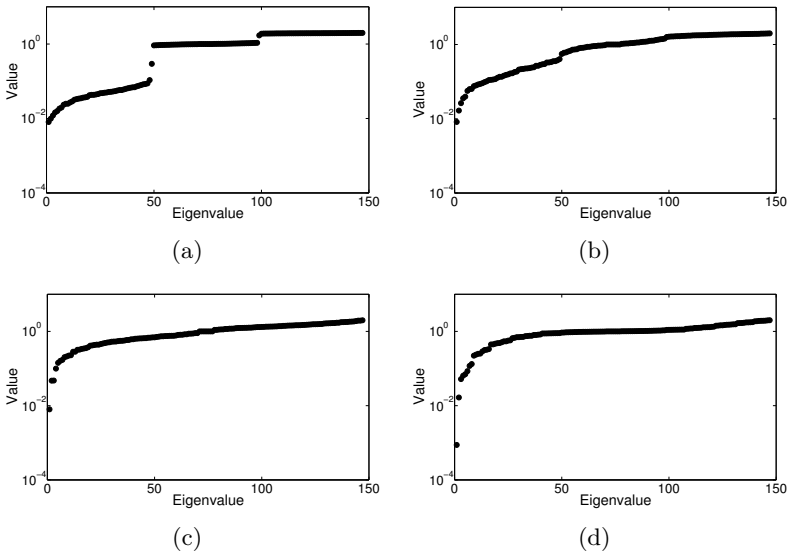


Figure 5-2.7: Spectra at $T = 4$ for the modified permeability field with high permeability (a) 10^1, (b) 10^2, (c) 10^3 and (d) 10^4.

In Figure (a), we see the same clustered pattern as in Figure 5-2.5. As we increase the permeability in the bottom and top layer, while keeping the middle layer constant at 10^0, extreme eigenvalues start to appear. In Figure (d), one eigenvalue is approximately 10^1 smaller than the second smallest eigenvalue. For a very large permeability difference, as illustrated in Figure 5-2.8, two extreme eigenvalues, corresponding to two jumps, are isolated from the main cluster.

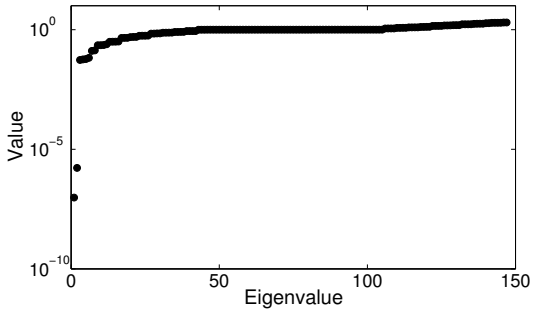


Figure 5-2.8: Spectrum at $T = 4$ for the modified permeability field with high permeability 10^8.

Although it remains unclear what the cause of the clustered eigenvalue pattern in Figure

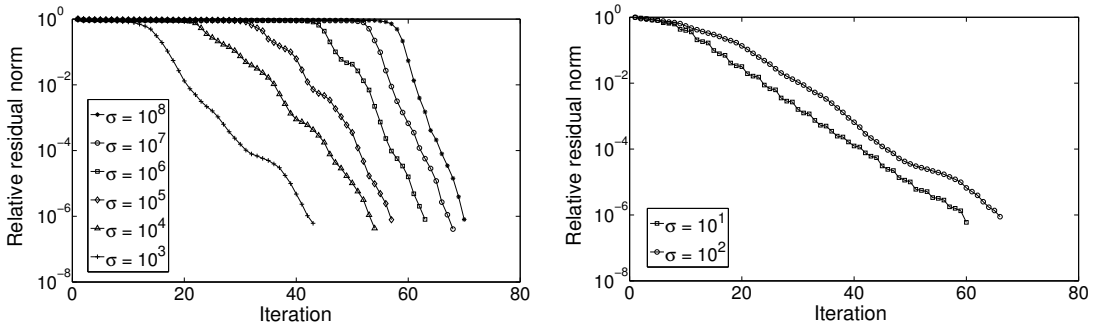
5-2.5 is, we can conclude that high permeability differences between the horizontal layers results in isolated extreme eigenvalues. As predicted in [11], we find that the number of extreme eigenvalues (two in this case) corresponds to the number of jumps. Furthermore, as shown next, the size of the extreme eigenvalues can be related to the size of the jumps.

Table 5-2.1 shows the size of the smallest eigenvalue for different choices of the high permeability in Figure 5-2.6.

low permeability	high permeability	smallest eigenvalue	2nd smallest eigenvalue	3rd smallest eigenvalue
10^0	10^1	$8.1 \cdot 10^{-3}$	$9.9 \cdot 10^{-3}$	$1.2 \cdot 10^{-2}$
10^0	10^2	$8.2 \cdot 10^{-3}$	$1.7 \cdot 10^{-2}$	$2.6 \cdot 10^{-2}$
10^0	10^3	$8.0 \cdot 10^{-3}$	$4.7 \cdot 10^{-2}$	$4.7 \cdot 10^{-2}$
10^0	10^4	$8.7 \cdot 10^{-4}$	$1.7 \cdot 10^{-2}$	$5.2 \cdot 10^{-2}$
10^0	10^5	$9.9 \cdot 10^{-5}$	$1.7 \cdot 10^{-3}$	$5.6 \cdot 10^{-2}$
10^0	10^6	$7.2 \cdot 10^{-6}$	$1.6 \cdot 10^{-4}$	$5.6 \cdot 10^{-2}$
10^0	10^7	$9.4 \cdot 10^{-7}$	$1.6 \cdot 10^{-5}$	$5.5 \cdot 10^{-2}$
10^0	10^8	$9.6 \cdot 10^{-8}$	$1.7 \cdot 10^{-6}$	$5.4 \cdot 10^{-2}$

Table 5-2.1: Smallest eigenvalue for different permeability jumps.

We denote the high permeability value as σ. As we have already seen in Figure 5-2.7, the smallest two eigenvalues are more or less constant and of the same order as the third smallest eigenvalue for $\sigma \leq 10^3$. For $\sigma > 10^3$, the smallest eigenvalue is proportional to the size of the jump. The second smallest eigenvalue is also proportional, but only for $\sigma > 10^4$. The third smallest eigenvalue remains constant for all jumps. As illustrated in Figure 5-2.8 for $\sigma = 10^8$, all but two eigenvalues are clustered around one. As the gap between the extreme eigenvalues and the main cluster increases, we expect from the discussion in Section 2-4-3 and Chapter 3 that, in general, GMRES will have more difficulties to solve the linear system. This is indeed the case, as demonstrated in the convergence plots of Figure 5-2.9. Note that the example at the start of Chapter 3 is in fact the SPE5 case with modified permeability and $\sigma = 10^6$.



(a) (b)

Figure 5-2.9: Convergence of (Jacobi) preconditioned GMRES(100), with σ = high permeability.

The cycle size is set to 100 to prevent a restart, and to allow the Ritz values to find the small eigenvalues. For $\sigma \geq 10^3$ in Figure (a), the duration of the stall in the GMRES residual convergence is approximately proportional to the size of the permeability jumps. For $\sigma = 10^8$, for example, GMRES starts to converge after 55 iterations, whereas GMRES for $\sigma = 10^4$ needs only 20 iterations. This theory does not hold for $\sigma < 10^3$, as shown in Figure 5-2.9(b). Given the number of iterations required to converge, the separated clusters of eigenvalues that we saw in Figure 5-2.7(a) and (b) are causing more difficulties for GMRES than the single isolated eigenvalue shown in Figure 5-2.7(c) and (d).

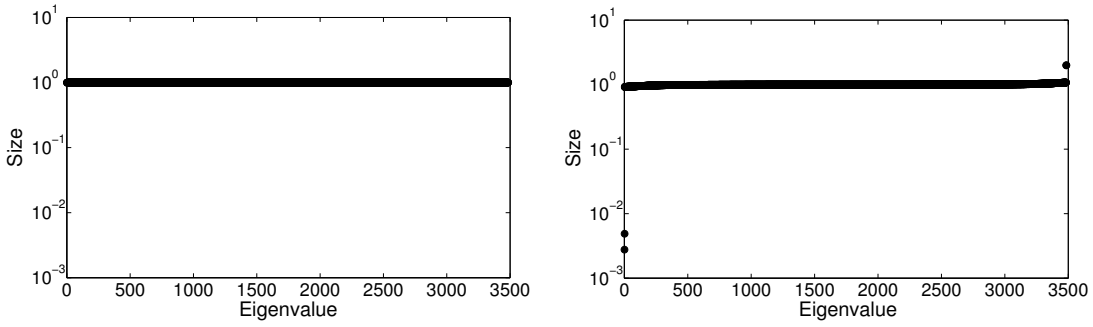
From the SPE5 spectral analysis, we conclude the following:

- The eigenvectors, in this case, resemble the flow pattern of water saturation. When the reservoir is saturated, the eigenvectors resemble the permeability field.
- The number of isolated eigenvalues is proportional to the number of permeability jumps between different layers.
- The size of the isolated eigenvalues is proportional to the size of the permeability jumps between different layers.
- Isolated extreme eigenvalues occur, in this case, only for permeability differences of order $\mathcal{O}(10^4)$ or larger.

5-2-3 SAGD-SMALL spectrum

So far, we have only considered the permeability jumps as a cause for isolated/extreme eigenvalues. The SAGD-SMALL case illustrates that it is also possible that strong nonlinearities caused by heating or sudden pressure changes can affect the spectrum. In Figure 5-2.10, the spectrum of the (diagonally scaled) SAGD-SMALL pressure matrix

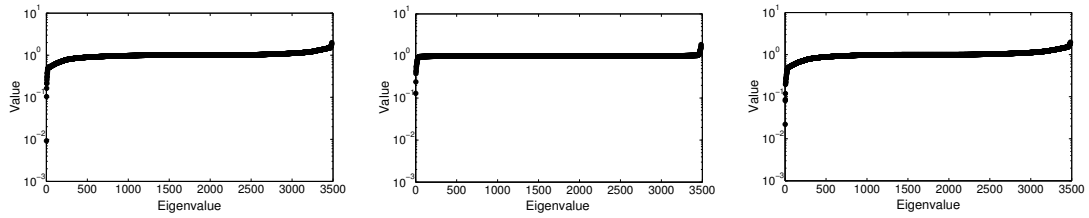
is plotted. Figure (a) shows the ordered eigenvalues before the heating is started, figure (b) shows the eigenvalues right after the heating is started.



(a) (b)

Figure 5-2.10: Spectrum of SAGD-SMALL pressure matrix before and after the heating is started.

Before the heating, all eigenvalues are clustered around one. After the heating is started, two extreme eigenvalues appear between 10^{-2} and 10^{-3}. For other time steps, the clustering of the spectrum and the number of isolated eigenvalues will vary, as shown in Figure 5-2.11.



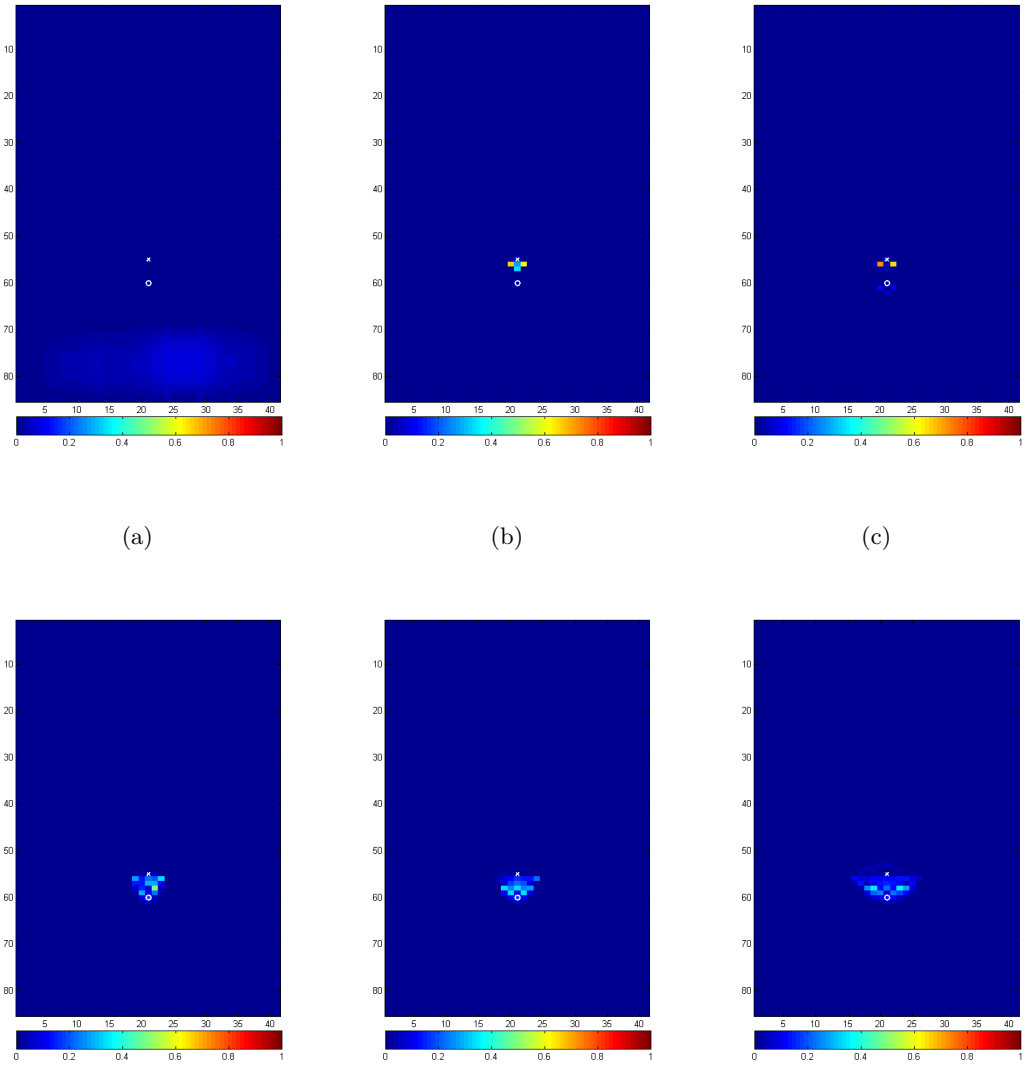
(a) (b) (c)

Figure 5-2.11: Spectrum of SAGD-SMALL pressure matrix at $T = 30, 40$ and 70.

The small eigenvalues are often isolated, but rarely as extreme as we have seen in the SPE5 case with modified permeability. This could be explained by the fact that the non-linearities (e.g. heating, steam injection) are not severe enough to spawn extreme eigenvalues in the pressure system. Furthermore, although there are clear jumps visible between the horizontal layers of the SAGD-SMALL permeability field (Figure 5-1.7), the jumps are often not larger than 10^3. Given the previous discussion on the SPE5 case, we believe that these permeability differences are not sufficient to produce extreme eigenvalues smaller than $\mathcal{O}(10^{-3})$. This hypothesis was confirmed by analyzing the spectrum of the SAGD-SMALL case for a large number of simulation times, a few of which are shown in Figure 5-2.10 and Figure 5-2.11. In none of the cases, eigenvalues

smaller than order $\mathcal{O}(10^{-3})$ were found. We will see in the upcoming sections, however, that it is still beneficial to apply deflation to the isolated eigenvalues, even though they are not as extreme as is often the case in deflation literature.

Figure 5-2.12 shows the eigenvectors of the SAGD-SMALL case at reporting times $T = 1$, $T = 10$, $T = 20$, $T = 50$, $T = 60$ and $T = 70$.



(a) (b) (c)

(d) (e) (f)

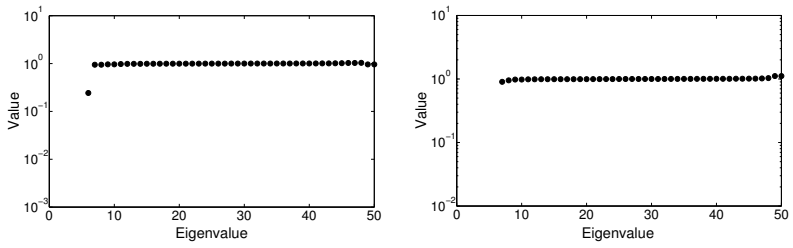
Figure 5-2.12: Absolute values of the eigenvectors for (a) $T = 1$, (b) $T = 10$, (c) $T = 20$, (d) $T = 50$, (e) $T = 60$ and (f) $T = 70$.

As in the SPE5 case, the eigenvector patterns of the SAGD-SMALL pressure matrix are similar to the saturation patterns around the injector and producer. In addition, observe in Figures (d), (e) and (f) that the top half of the shape around the injector and producer is somewhat flattened. A layer of low permeability in this region is most likely affecting the saturation, which in turn influences the eigenvector. Apart from a small region around the injector and producer, the eigenvector values are close to zero. Therefore, we conclude that for deflation to be effective it is important to have accurate eigenvector approximations mainly in the regions where flow occurs.

5-2-4 Effect of the (parallel) preconditioner

Remark 5-2.2. *In the upcoming spectral plots, negative eigenvalues were ignored, and for imaginary eigenvalues only the real part is shown. In addition, the eigenvalues around* 10^{-15} *are interpreted as zero [12, p. 65] and omitted from the figures.*

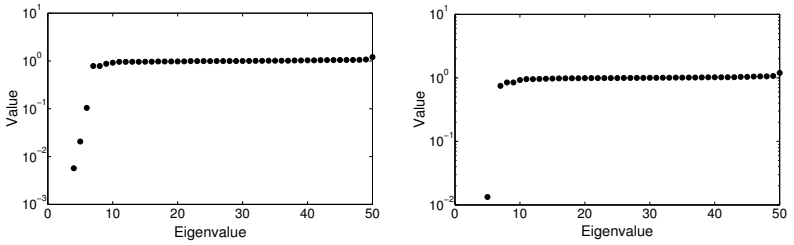
To assess the potential of deflation, we are interested in the action of the AMG preconditioner on the pressure matrix. To avoid having to implement AMG in Matlab, we approximate the eigenvalues directly in IX using the harmonic Ritz approximations from Section 3-4-3. The cycle size is chosen sufficiently large (e.g. $m = 100$) to ensure that the approximations are accurate, while the number of harmonic Ritz values is set relatively high (50) to ensure full coverage of the spectrum. The result is shown in Figure 5-2.13 for the linear system resulting from non-linear iteration three and five.



(a) (b)

Figure 5-2.13: Spectrum of the CPR-preconditioned (AMG & ILU) SAGD-SMALL pressure matrix at non-linear iteration three and five.

For both linear systems, the preconditioning is effective. In non-linear iteration three, all but one eigenvalues are clustered around one. The single isolated eigenvalue is not extreme. In non-linear iteration five, all eigenvalues are clustered around one. To provide a reference for the AMG preconditioner, Figure 5-2.14 shows the spectra for the case where AMG is replace by diagonal scaling as the pressure system preconditioner (similar to the spectra in Section 5-2-3).



(a) (b)

Figure 5-2.14: Spectrum of the CPR-preconditioned (Jacobi & ILU) SAGD-SMALL pressure matrix at non-linear iteration three and five.

In this case, the preconditioning is less effective. Several eigenvalues are isolated from the main cluster for non-linear iteration three, and for non-linear iteration five one eigenvalue is nearly extreme. Furthermore, although it is not directly visible in the figure, the cluster of eigenvalues is more skewed away from one for Jacobi compared to AMG. Because AMG is clipping most of the spectrum to one, whereas Jacobi leaves behind several isolated eigenvalues, we can expect deflation to be more effective when combined with Jacobi. We will see in the upcoming section on harmonic Ritz deflation that this is indeed the case.

So far, only the pressure matrix has been analyzed. To investigate the effect of the parallel implementation, we use the harmonic Ritz eigenvector approximations to approximate the 50 smallest eigenvalues and corresponding eigenvectors of the CPR-preconditioned reservoir matrix. We verify the hypothesis that the ILU preconditioner, which is applied to the reservoir matrix, increasingly behaves like a Jacobi preconditioner as the number of parallel subdomains is increased. The change in behavior can be attributed to the fact that connections between parallel subdomains are ignored, in order to reduce the fill-in. The spectrum of the reservoir matrix with a CPR stage one Jacobi and stage two ILU preconditioner is given in Figure 5-2.15.

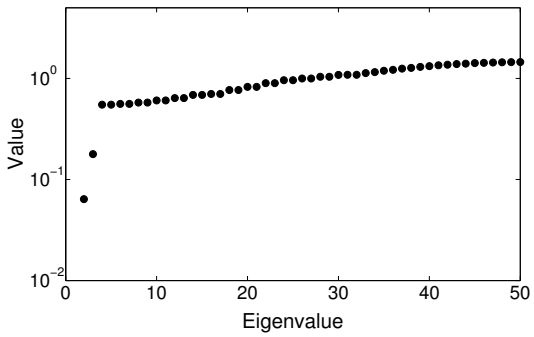
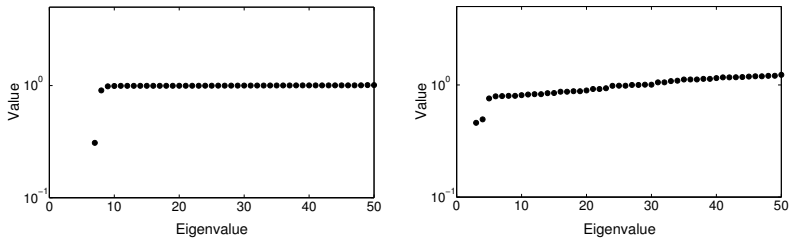


Figure 5-2.15: Spectrum of the CPR-preconditioned (Jacobi & ILU) SAGD-SMALL reservoir matrix.

In Figure 5-2.16, we plot the spectra for the CPR-preconditioned reservoir matrix (with AMG), using either one or eight processors.

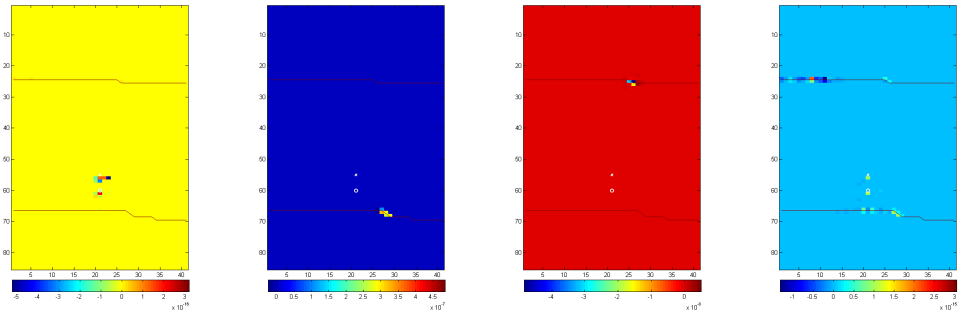


(a) (b)

Figure 5-2.16: Spectrum of the CPR-preconditioned (AMG & ILU) SAGD-SMALL reservoir matrix with (a) $n_p = 1$ and (b) $n_p = 8$.

In Figure 5-2.16(a), most of the spectrum is clustered around one, similar to the preconditioned pressure matrix in Figure 5-2.13. In Figure (b), however, the spectrum is slanted away from zero, much like the spectrum in Figure 5-2.15. Intermediate choices of n_p will result in less slanting, while the spectra for $n_p > 8$ will converge further towards the eigenvalues in Figure 5-2.15. This confirms the idea that the parallel implementation of ILU negatively affects the spectrum.

Lastly, we are interested in the effect of the parallel ILU implementation on the eigenvectors. In particular, we investigate if the negative impact on the spectrum can be remedied using deflation. The eigenvectors corresponding to the smallest eigenvalues of the CPR-preconditioned reservoir matrix are shown in Figure 5-2.17. Two parallel subdomains are used, of which the boundaries are shown in the figure. Note that the first processor is assigned two regions (at the top and at the bottom).



(a) (b) (c) (d)

Figure 5-2.17: Several eigenvectors corresponding to the smallest eigenvalues of the CPR-preconditioned SAGD-SMALL reservoir matrix with $n_p = 2$.

Similar to the eigenvectors in Figure 5-2.12, non-zero values occur around the injector and the producer in Figure 5-2.17(a). When the preconditioner is used in parallel instead of serial, however, non-zero values also occur precisely on the edges of the the parallel domains, as shown in Figures (b), (c) and (d). In fact, although (due to the scale) it is not visible in this figure, non-zero values were found along almost all of the boundaries. This observation gave rise to the implementation of deflation vectors with ones on the edges of the parallel subdomains, in hope of approximating the eigenvectors. Results are briefly discussed in Section 5-5-2.

5-2-5 Summary of findings

Using the BO, SPE5 and SAGD-SMALL cases, we draw a number of conclusions regarding the occurrence of isolated/extreme eigenvalues:

- Isolated, and especially extreme, eigenvalues harm the convergence of GMRES. If the corresponding eigenvectors can be approximated (using either harmonic Ritz approximations or a physics-based approach), then deflation is a promising method to speed up convergence.
- The number of isolated (extreme) eigenvalues is proportional to the number of permeability jumps, and the size of the isolated (extreme) eigenvalues is proportional to the size of the permeability jumps.
- In both the SAGD-SMALl and SPE5 case, no extreme eigenvalues occur for permeability jumps of order $\mathcal{O}(10^3)$ or smaller.
- In the SAGD-SMALL case, temperature effects negatively affect the spectrum.
- AMG is more effective than diagonal scaling as the first stage preconditioner in CPR, as demonstrated by the quality (i.e. number of isolated/extreme eigenvalues) of the spectrum.
- For increasing n_p, the ILU preconditioner will behave more like diagonal scaling.

In the analysis, we find several ideas for alternative deflation algorithms:

- In the SPE5 and SAGD-SMALL cases, the eigenvector non-zero pattern shows similarities with the saturation. Results of a deflation algorithm using the saturation instead of the permeability are discussed in Section 5-5-1.
- When the ILU preconditioner is used in parallel, eigenvector values appear on the boundaries of the parallel subdomains. Results of a deflation algorithm attempting to approximate these eigenvectors are discussed in Section 5-5-2.

5-3 Harmonic Ritz deflation

In this section, we discuss the results obtained from applying harmonic Ritz deflation to the BO, SPE5 and SAGD-SMALL cases. The implementation in Algorithm 5 is used both in Matlab an IX. In the Matlab experiments, we compare the harmonic Ritz results to deflation using exact eigenvectors and GMRES without deflation. In IX, harmonic Ritz deflation is only compared to GMRES without deflation, as it is not possible to compute the exact eigenvectors. We start our experiments using $A_p = AM^{-1}$, but also show that it is possible to take $A_p = A$.

Remark 5-3.1. *Deflated GMRES is denoted as DGMRES(m,d), where m is the cycle size and d is the number of deflation vectors used. Unless noted otherwise, we assume that the deflation vectors correspond to the smallest d (approximated) eigenvalues.*

5-3-1 Matlab experiments

In most of the experiments in this section, a simple Jacobi preconditioner is used, applied from the right. As discussed in the motivation at the start of Chapter 3, our goal is to apply deflation artificially in order to achieve fast convergence from the start of the simulation. To illustrate the potential of removing harmful eigenvalues, we first apply deflation using exact eigenvectors. The eigenvectors of the pressure matrix are computed using the Matlab function eig, applied to AM^{-1}.

In Figure 5-3.1, the convergence history of the SPE5 case with modified permeability and $\sigma = 10^6$ (i.e. Figure 3-1.1) is appended with:

- **DGMRES(20,1) exact**. Deflated GMRES with cycle size 20 and one exact eigenvector, i.e. $Z \in \mathbb{R}^{n_c \times 1}$.
- **DGMRES(20,2) exact**. Deflated GMRES with cycle size 20 and two exact eigenvectors, i.e. $Z \in \mathbb{R}^{n_c \times 2}$.

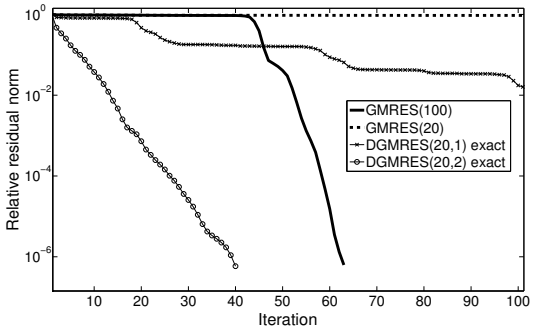


Figure 5-3.1: (D)GMRES(20) residual convergence for the SPE5 case with modified permeability and $\sigma = 10^6$.

Recall from Table 5-2.1 that there are two extreme eigenvalues in the spectrum of the pressure system. We make two observations from the figure above. Firstly, deflating only the smallest eigenvalue marginally improves the convergence of GMRES(20). The second extreme eigenvalue still cannot be found by the Ritz values in twenty iterations. Secondly, if we deflate the two smallest eigenvalues, the convergence of GMRES(20) significantly improves. Having deflated these eigenvalues, only the cluster of eigenvalues clost to one remains. As shown by the convergence history in Figure 3-1.2, these eigenvalues can be found by the Ritz values within twenty iterations.

In Figure 5-3.2, the SPE5 convergence history is repeated, this time with $m = 100$ for DGMRES.

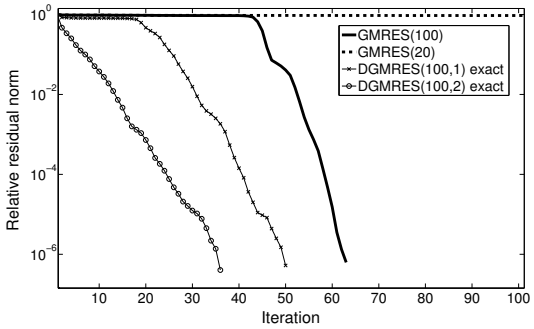


Figure 5-3.2: (D)GMRES(100) residual convergence for the SPE5 case with modified permeability and $\sigma = 10^6$.

Because we allow the Ritz values to converge to the second smallest eigenvalue, the residual reduction for DGMRES(100,1) is better than the residual reduction for DGMRES(20,1). Comparing the two previous figures, we also observe that the DGMRES(100,2) and DGMRES(20,2) are nearly equal. Furthermore, the convergence speed of GMRES(100) (after the stall) and DGMRES(100,2) (from the start) is comparable. As said, twenty iterations is sufficient for the Ritz values to converge towards all other small eigenvalues. In terms of the convergence speed, the potential of deflation is clearly illustrated. For GMRES(100), fast convergence occurs only after the Ritz values have converged to the extreme eigenvalues (natural deflation). We obtain the same fast convergence from the start by deflating the two extreme eigenvalues artificially.

Recall from Figure 5-2.1 that the spectrum of the BO case has two isolated eigenvalues. The rest of the spectrum is clustered between 10^{-2} and 10^{1}. In Figure 5-3.3, GMRES(20) and GMRES(100) are compared with DGMRES(20) for a varying number of deflation vectors.

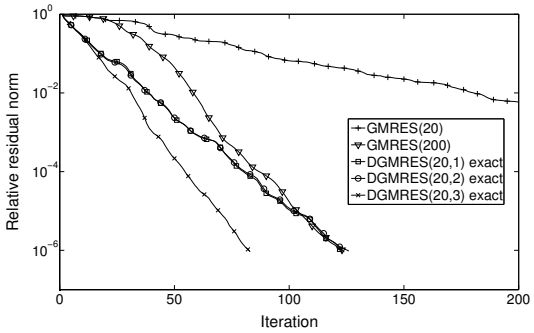


Figure 5-3.3: (D)GMRES residual convergence for the BO case.

Using DGMRES(20,1), only the smallest eigenvalue is deflated from the linear system. Even though this results in faster convergence in the beginning, after about 100 iterations GMRES(100) and DGMRES(20,1) reach the same residual norm. Surprisingly, deflating the second smallest eigenvalue does not improve convergence. Only when the third smallest eigenvalue is also deflated, do we achieve a noticeable improvement. Since the left tail of the eigenvalue cluster contains many eigenvalues away from zero (i.e. $\mathcal{O}(10^{-2})$), and the second smallest eigenvalue is only somewhat isolated, other error characteristics (such as the location in the domain) may explain this behavior. In our experience, the convergence after deflating additional eigenvalues (on top of the extreme eigenvalues) may not always improve, but at least it will never deteriorate.

Having demonstrated the potential of deflation using exact eigenvectors, we now discuss the results of deflated GMRES using harmonic Ritz eigenvector approximations. In Figure 5-3.4, we plot the linear solve of the BO case using GMRES(20), DGMRES(20) using one exact eigenvector or DGMRES(m) using harmonic Ritz (HR) vectors with varying m.

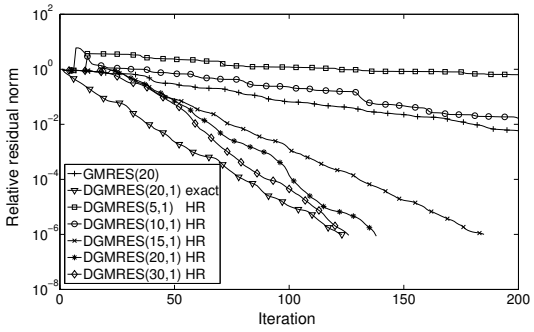


Figure 5-3.4: (D)GMRES residual convergence for varying m.

The fastest convergence is achieved by DGMRES(20,1) with one exact eigenvector. DGMRES(30,1) using one harmonic Ritz vector has higher residual norms in the first 100 iterations, but reaches the tolerance of 10^{-6} at nearly the same iteration count.

For lower values of m, the convergence is slower, although still significantly faster than GMRES without deflation. For $m = 10$ and $m = 5$, however, the harmonic Ritz vector is no longer a sufficiently accurate approximation of the true eigenvector. As a result, DGMRES is slower than GMRES. To illustrate this phenomenon, the exact eigenvector and harmonic Ritz eigenvector approximation after five and thirty iterations are plotted in Figure 5-3.5.

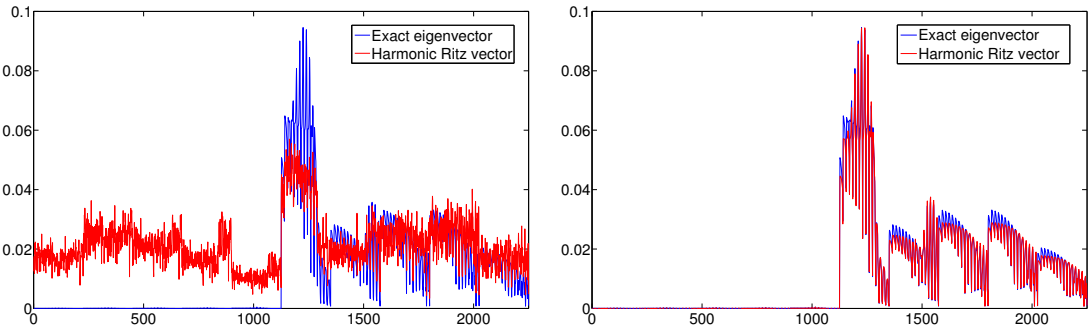


Figure 5-3.5: The harmonic Ritz vector and the exact eigenvector after (a) five and (b) thirty iterations.

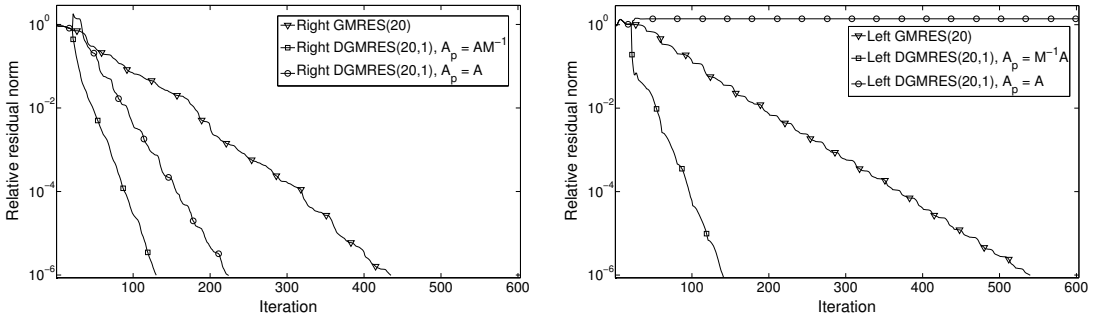
After five iterations, the harmonic Ritz vector is not a good approximation of the true eigenvector. As shown in Figure 5-3.4, deflation is not effective in this case. After thirty iterations, on the other hand, the harmonic Ritz vector and the eigenvector nearly overlap and deflation significantly improves the convergence of GMRES. For intermediate choices of m, the approximation will be more/less accurate, as demonstrated in the convergence history in Figure 5-3.4.

After repeating the above experiment for the SPE5 case with modified permeability, we find that the size of the permeability jump imposes a requirement on the cycle size. If we run the simulation with $\sigma = 10^8$, for example, it turns out that $m = 20$ is not sufficient to obtain sufficiently accurate harmonic Ritz eigenvector approximations. As a result, DGMRES will not converge. For this particular case, $m \geq 40$ is required for deflation to be effective. We believe that this correlation is caused by the extreme eigenvalues in the SPE5 case, which become more isolated and extreme as we increase σ (Table 5-2.1). The more extreme the eigenvalues become, the longer it takes for the smallest Ritz values to converge to the extreme eigenvalues. Therefore, more iterations are required in a cycle to find decent approximation of the eigenvectors.

So far, we have been using deflation in combination with a Jacobi preconditioner, applied from the right. Furthermore, $A_p = AM^{-1}$ (in Algorithm 5 and 6) was used in all previous plots. The (AMG) preconditioner in IX is one of the bottlenecks in the linear solve. Therefore, deflation is relatively expensive if the preconditioner has to be applied multiple times in the application of P_1 and P_2. To limit the deflation overhead, we

would prefer to take $A_p = A$ instead. Theoretically, it is possible that the harmful eigenvalues that deflation acts on are precisely the eigenvalues that the preconditioner cannot resolve. Therefore, deflation applied to $A_p = A$ might be just as effective as applying deflation to $A_p = AM^{-1}$.

The convergence for DGMRES(20,1) using one harmonic Ritz vector, a right Jacobi preconditioner and $A_p = A$, is shown in Figure 5-3.6(a). Although only right-preconditioning is relevant for our purposes, from a theoretical point of view it is also worth investigating the choice $A_p = A$ for left-preconditioning. The result is shown in Figure 5-3.6(b).



(a) (b)

Figure 5-3.6: Convergence of (a) left- and (b) right-preconditioned (D)GMRES for the BO case.

For right-preconditioning, using $A_p = A$ instead of $A_p = AM^{-1}$ causes slower convergence, although DGMRES(20,1) is still faster than GMRES(20). For left-preconditioning, using $A_p = AM^{-1}$ results in approximately the same convergence as in the right-preconditioned case. For $A_p = A$, however, the residual increases in the first fifty iterations, after which GMRES stalls. No satisfying explanation for this phenomenon was obtained, but we suspect that the answer might be found in the different linear systems: left-preconditioned GMRES solves $M^{-1}Ax = M^{-1}b$ and right-preconditioned GMRES solves $AM^{-1} = b$. We remark that, to our knowledge, the formulation of the deflation operators P_1 and P_2 for preconditioned linear systems, as well as the choice of $A_p = A$ has barely received any attention in relevant literature. The authors in [12] and [80] discuss the use of deflation combined with preconditioning, but do not treat the previous two topics. As argued before, these choices are crucial for the performance in practical applications.

In the previous plot, we find that $A_p = A$ can be used with right-preconditioning in DGMRES while still converging faster than GMRES without deflation. Note that there is a slight residual increase of DGMRES(20,1) ($A_p = A$) in Figure 5-3.6(a). The jump appears in iteration 21, which is the first iteration after the restart. At this point, deflation using $A_p = A$ introduces solution components that temporarily increase the residual. We risk ending our simulation for a larger residual than the initial residual.

After a number of numerical experiments, we find that the following measures can be used to reduce the temporary increase of the residual:

- Increase the value of m. The residual increase after the restart becomes smaller, if the harmonic Ritz vectors are allowed more time to converge towards to the exact eigenvectors. If exact eigenvectors are used (with $A_p = A$), the residual will never increase.
- Use a stronger preconditioner. If an ILU preconditioner is used instead of Jacobi, the residual increase will be smaller.

Despite the results in Matlab above, we have not encountered any issues with the residual increase in our IX experiments.

Lastly, before we continue with the IX results, we compare the following two deflation methods:

1. Apply the harmonic Ritz deflation method once after a full cycle of GMRES, and freeze the deflation vectors for consecutive cycles. This is the approach that we have used so far.
2. After each cycle, apply the harmonic Ritz deflation method and append the deflation vectors to the previous deflation vectors.

As shown in Figure 5-3.7, the second approach is slightly more efficient.

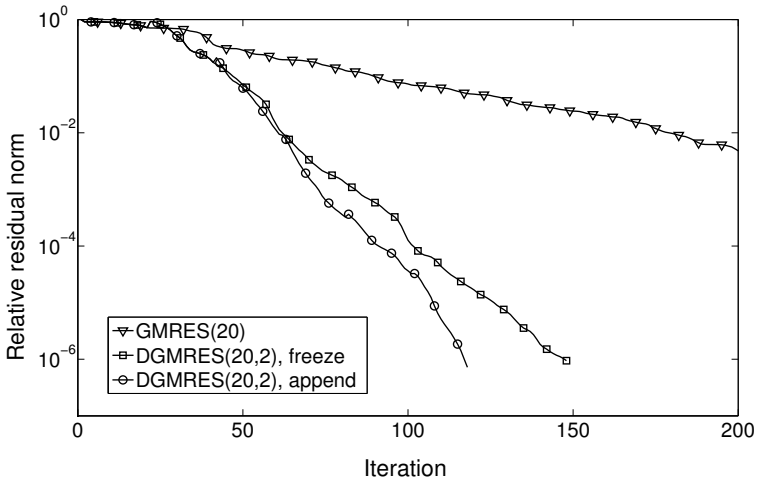


Figure 5-3.7: Comparison of freezing the deflation vectors versus appending the deflation vectors in the BO case.

Up to 40 iterations, both choices are approximately equal. As the simulation continues, appending deflation vectors further improves convergence. When the tolerance of 10^{-6} is reached, the freezing method is roughly 30 iterations faster. In experiments with other cases, we observe similar or better convergence improvements if deflation vectors are

appended instead of frozen. Repeatedly computing and appending the harmonic Ritz vectors, however, has two disadvantages. Firstly, the memory requirements increase as the size of Z increases. Secondly, the computational costs of the matrix-vector products, inner products and Galerkin solve (in applying P_1 and P_2) increase as well. These costs do not outweigh the time gain from the reduced number of iterations. In conclusion, even though appending deflation vectors can improve convergence, we will mainly use the freezing method.

5-3-2 IX experiments

Having demonstrated the potential of the harmonic Ritz deflation method in Matlab, we will continue with the results from IX. Each simulation in this section uses $A_p = A$ and preconditioning applied from the right. All simulations are run in serial. We will use the SPE5 case with modified permeability and $\sigma = 10^8$, as well as the SAGD-SMALL case. In the previous section, we showed that harmonic Ritz deflation can effectively eliminate the harmful eigenvalues and thereby improve convergence. As will become clear in this section, however, the performance in IX is not as good and the overhead of deflation is too high to compete with the existing CPR preconditioning scheme.

The following variables are used to compare the results:

- **Non-linears**. The amount of non-linear iterations in the simulation.
- **Fails**. The amount of failed non-linear iterations.
- **Outer linears**. The amount of iterations used to solve the linear systems generated by the non-linear iterations.
- **Inner linears**. The amount of iterations used to solve the pressure systems generated by the second stage of the CPR preconditioning in the outer GMRES loop.
- **CPU time**. The overall CPU time (in seconds) of the linear solve.

We refer to the linear solve of the full linear system and the pressure solve as the outer linear solve and the inner linear solve, respectively. For each outer linear iteration, there is at least one inner linear iteration. Note that the number of non-linears is equal to the number of linear systems that need to be solved in the outer linear solve. Similarly, the number of outer linears is equal to the number or pressure systems that need to be solved in the inner linear solve. We compare the overall CPU time of the linear solve to analyze the overhead of deflation.

The default settings of the IX pressure solve are summarized below.

Setting	Default value
Pressure solve tolerance	10^{-6}
Cycle size	20
Minimum number of iterations	1
Maximum number of iterations	1
Preconditioner	AMG

Table 5-3.1: Default settings in IX.

Since we only apply deflation to the pressure solve in this section, the tolerance of the outer linear solve and the non-linear solve are omitted. In the first numerical experiment, we use the following settings:

Setting	Default value
Pressure solve tolerance	10^{-6}
Cycle size	30
Minimum number of iterations	30
Maximum number of iterations	60
Preconditioner	Jacobi

Table 5-3.2: Settings for the results in 5-3.3.

By default, IX only uses a single iteration of AMG-preconditioned GMRES. In order to use harmonic Ritz deflation, a restart is required. Therefore, we set the minimum number of iterations equal to the cycle size. By using a weaker preconditioner (Jacobi), we highlight the advantage of deflation. The cycle size is chosen sufficiently large to guarantee accurate eigenvector approximations. In general, our experience is that m should be at least 20. Similar to the Matlab experiments, we find that appending the deflation vectors in IX does not improve the convergence enough to justify the increased memory requirements and CPU time. Therefore, the deflation vectors are frozen after the first cycle.

Table 5-3.3 shows a comparison of GMRES(30) and DGMRES(30,3) in IX, using the settings from Table 5-3.2 and the SPE5 case with modified permeability ($\sigma = 10^8$).

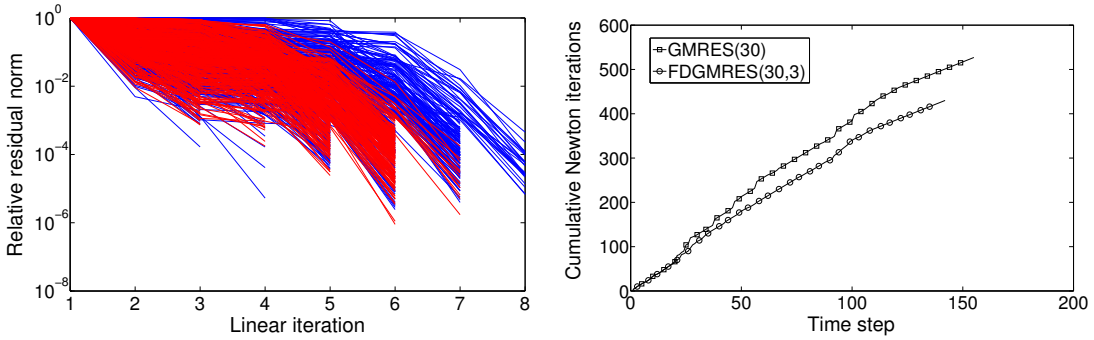
	GMRES(30)	DGMRES(30,3)
Non-linears	527	430
Fails	0	0
Outer linears	$2,057$	$1,606$
Inner linears	$123,420$	$96,360$
CPU time	6.58	5.68

Table 5-3.3: Comparison of GMRES and DGMRES(30,3) in IX.

Firstly, note that $2,057 \times 60 = 123,420$ and $1,606 \times 60 = 96,360$, which implies that neither GMRES(30) nor DGMRES(30,3) reaches the tolerance of 10^{-6} in any of the pressure solves. The number of outer linears is reduced by approximately 22% because the residual in each pressure solve is (slightly) better for DGMRES compared to GMRES. As a result, the outer linear solve will converge faster. On average, GMRES(30) uses $2,057/527 = 3.90$ outer linears per non-linear, and DGMRES(30,3) uses $1,677/456 = 3.73$ outer linears per non-linear. This is an improvement of about 5%.

The CPU time of the linear DGMRES solve is 0.9 seconds smaller than the linear solver time for GMRES. On average, GMRES and DGMRES require respectively $6.58/2,057 = 3.2 \cdot 10^{-3}$ and $5.68/1,606 = 3.5 \cdot 10^{-3}$ seconds per pressure solve. Hence, the overhead of deflation, for this particular case, is equal to $3.5 \cdot 10^{-3} - 3.2 \cdot 10^{-3} = 3.0 \cdot 10^{-4}$ seconds per pressure solve. We conclude that the overhead of deflation is small enough to render the method efficient.

Figure 5-3.8 contains a combined plot of all linear solves (of the full system) and a plot of the cumulative Newton iterations.



(a) (b)

Figure 5-3.8: Linear solves and cumulative Newton iterations of the SPE5 case..

In Figure (a), we see that the outer linear solves in DGMRES (red lines) use (on average) less iterations to converge, while the residual norm is often smaller. The number of

non-linear iterations to solve a particular timestep is smaller for DGMRES, as illustrated in Figure (b). From Table 5-3.3, we know that GMRES(30) requires 527 non-linear iterations to complete the simulation, while DGMRES(30,3) needs only 430. In general, our experience is that a lower outer linears count does not necessarily translate into less non-linear iterations. The tolerance of the outer linear solve is equal for both GMRES(30) and DGMRES(30,3). Therefore, as long as no fails occur and the maximum number of iterations (of the outer linear solve) is high enough we reason that, in theory, the non-linear iteration count should be equal for GMRES and deflated GMRES. The fluctuation might be caused by the implementation of the non-linear solver in IX, which is beyond the scope of this thesis.

Although the speedup is not as significant as was shown in the Matlab experiments (e.g. Figure 5-3.1), deflation in the diagonally scaled IX pressure system is still efficient. In the next experiment, we compare deflation to the AMG preconditioner. Table 5-3.4 compares the results from Table 5-3.3 to GMRES(30) with an AMG preconditioner.

	DGMRES(30,3) (Jacobi)	GMRES(30) (AMG)
Non-linears	430	395
Fails	0	0
Outer linears	1,606	1,099
Inner linears	96,360	66,59
CPU time	5.68	1.22

Table 5-3.4: Comparison of DGMRES(30,3) (with Jacobi) and GMRES(30) (with AMG) in IX.

The main difference between the two methods is the amount of inner linears. By inspecting the convergence history, we find that AMG reaches the tolerance in the pressure solve after (on average) $6,659/1,099 \approx 6$ inner linear iterations. Jacobi-preconditioned GMRES combined with deflation never reaches the tolerance, since $96,360/1,606 = 60$ (the maximum number of iterations). Consequently, DGMRES with Jacobi requires over 14 times as many inner iterations, and is 4.7 times slower in terms of CPU time than GMRES with AMG. Moreover, in the default IX settings the pressure solve uses only one AMG-preconditioned GMRES iteration. Our experiments show that the residual in this pressure solve setup is often sufficient to guarantee good convergence of the outer linear solve. When only one iteration of GMRES is used, the CPU time of the linear solve will be a fraction of the 1.22 seconds above.

For this experiment, it is not possible to use DGMRES with an AMG preconditioner. As noted before, at least 20 iterations are required to obtain sufficiently accurate approximations of the eigenvectors. AMG-preconditioned GMRES, however, reaches machine-precision residual norms within 10 to 15 iterations. The physics-based deflation method in the next section is more suitable to be combined with AMG, because the physics-based

deflation vectors are applied from the start of the simulation. Our experiments for this deflation method show that adding deflation to AMG-preconditioned GMRES does not improve convergence. We believe that AMG is already very efficient in tackling the harmful eigenvalues, as demonstrated in the comparison of Figure 5-2.13 and 5-2.14. In section 2-5-1, we linked the occurrence of extreme eigenvalues to 'algebraically smooth' error nodes. AMG is very efficient in reducing these error characteristics, which explains why adding deflation does not improve convergence.

We present the results of two more experiments, both using the SAGD-SMALL case and a Jacobi preconditioner applied from the right. In an attempt to reduce the overhead of deflation, we reduce the cycle size to 20. The minimum number of iterations is also set at 20, while the maximum number of iterations is kept at 60. We know from our simulations that the water injection in the SAGD-SMALL case causes issues in the IX simulation. Shortly after the water injection starts, the non-linear time step size is reduced several times before the simulation can continue. Because isolated eigenvalues may be responsible (as seen Figure 5-2.10), we switch on deflation when the water injections starts. To reduce the computational costs, we do not use deflation for all previous time steps. The result is shown in Table 5-3.5 as 'switched' DGMRES (sDGMRES). A comparison is made with GMRES(20) using a Jacobi preconditioner, DGMRES(20,3) using a Jacobi preconditioner but without the switching and GMRES(20) with an AMG preconditioner.

	GMRES(20) (Jacobi)	DGMRES(20,3) (Jacobi)	sDGMRES(20,3) (Jacobi)	GMRES(20) (AMG)
Non-linears	291	291	291	291
Fails	0	0	0	0
Outer linears	929	937	935	933
Inner linears	44,322	36,676	38,701	55,54
CPU time	13.70	14.82	14.82	7.97

Table 5-3.5: Comparison of (switched) DGMRES with GMRES and AMG in IX.

Compared to GMRES(20) with a Jacobi preconditioner, DGMRES(20,3) reduces the number of pressure iterations (per non-linear iteration) by 17%. This is a significant improvement over the 5% reducation in the SPE5 case. The overhead of deflation, however, is larger, most likely due to the increased size of the SAGD-SMALL case ($12,145,225$ matrix elements versus $21,609$ elements for SPE5). Consequently, the CPU time of the linear solve is now higher for deflated GMRES. If we leave deflation off up to the point that the water injection starts, then the amount of inner linears increases by $38,701 - 36,676 = 2,025$ iterations. By coincidence, the CPU time gained by switching off deflation before the water injection is precisely compensated for by the increased number of inner linears. Lastly, GMRES with AMG is again significantly faster and more efficient than deflated GMRES with a Jacobi preconditioner.

In the last experiment, we vary the number of deflation vectors. The same case and settings are used as in the previous experiment. The results are shown in Tables 5-3.6 and 5-3.7.

	GMRES(20)	sDGMRES(20,1)	sDGMRES(20,3)	sDGMRES(20,5)
Non-linears	291	291	291	291
Fails	0	0	0	0
Outer linears	929	934	935	934
Inner linears	44,322	42,793	38,701	42,793
CPU time	13.70	15.29	14.82	15.12

Table 5-3.6: Switched DGMRES for a varying number of deflation vectors in IX.

	sDGMRES(20,10)	sDGMRES(20,15)	sDGMRES(20,20)
Non-linears	277	277	278
Fails	0	0	0
Outer linears	907	911	951
Inner linears	33,363	32,244	31,671
CPU time	15.00	16.40	17.85

Table 5-3.7: Switched DGMRES for a varying number of deflation vectors in IX.

The amount of inner linears decreases as we increase the number of deflation vectors, with one exception (sDGMRES(20,5)). The amount of non-linears and outer linears fluctuates. Most importantly, the CPU time of the linear solve increases for a larger number of deflation vectors. We conclude that the time gain from the decreased number of inner linears does not outweigh the overhead of deflation.

Lastly, we note that we have also applied harmonic Ritz deflation to the full reservoir system. We find that, unless the second stage ILU preconditioner is replaced by Jacobi, the outer linear solve rarely reaches 20 iterations. Even if we use Jacobi as both the first and second stage preconditioner, harmonic Ritz deflation does not improve convergence enough to compensate for the increased computational costs.

5-3-3 Summary of findings

In our Matlab experiments, we observe the following:

- Artificial deflation using harmonic Ritz vectors can significantly improve the performance of GMRES. The convergence speed will be approximately equal to the convergence speed attained after natural deflation of the extreme eigenvalues.

- The best performance gain is achieved by deflating all extreme eigenvalues. Deflating other eigenvalues does not necessarily improve convergence, although the convergence will never deteriorate.

- For most cases, the harmonic Ritz eigenvector approximation requires about 20 iterations to reach sufficient accuracy to be effective as a deflation vector. For very large permeability jumps, such as in the SPE5 case with modified permeability and $\sigma \geq 10^4$, we often need $m > 20$.

- In right-preconditioning, $A_p = A$ can be used in DGMRES instead of $A_p = AM^{-1}$ while still improving performance compared to GMRES without deflation.

- The residual jump after a restart (caused by taking $A_p = A$) can be remedied by increasing m or by using a stronger preconditioner such as AMG.

- Even though appending deflation vectors can reduce the iteration count, the method is too expensive. Instead, we freeze the deflation vectors after the first cycle.

In the IX experiments, we present one example (SPE5 case with modified permeability) where harmonic Ritz deflation reduces the number of pressure iterations enough to compensate for the overhead in terms of CPU time. The percental improvement in terms of outer linears per non-linear is rather small. For the second case (SAGD-SMALL), the percental decrease in outer linears is much larger. The results for this case also show, however, that the overhead of deflation is higher for a bigger pressure matrix. In particular, the overhead of deflation in the SAGD-SMALL is larger than the time gain from the reduced number of inner linears, rendering DGMRES inefficient.

Furthermore, DGMRES, which is combined with a Jacobi preconditioner in our experiments, is considerably slower than GMRES with an AMG preconditioner. We reason that AMG is efficient in tackling the error corresponding to the extreme eigenvalues. Consequently, adding deflation to AMG does not improve the performance of the pressure solve. Lastly, we attempt to reduce the overhead of deflation by switching deflation on/off during different stages of the simulation, and by varying the number of deflation vectors. Neither approaches leads to significant improvements. Therefore, we formulate the following two (related) objectives for an improved deflation method in IX:

- The overhead of harmonic Ritz deflation offsets the time gain from the improved convergence. Therefore, we require a cheaper deflation method.

- The harmonic Ritz deflation vectors do not speed up convergence enough to offset the overhead. Therefore, we require more effective deflation vectors.

This conclusion led to the development of a physics-based deflation method. The advantages of this method have been briefly highlighted in Section 4-3, but will be repeated in the next section along with the results.

5-4 Physics-based deflation

In physics-based deflation, we approximate the eigenvectors using the underlying physics. As discussed in Section 3-4-4, the span of the eigenvectors corresponding to the extreme eigenvalues in linear systems with strong heterogeneity can be approximated by the span of a set of physics-based deflation vectors. In our applications, the permeability is generally responsible for the largest jumps in the coefficients of the reservoir equations. The deflation vectors can be constructed manually, or computed automatically using the subdomain-levelset method described in Section 4-2. The main advantages of physics-based deflation are:

- The deflation vectors are defined a priori. Therefore, harmful eigenvalues can be eliminated from the spectrum from the start of the linear solve.
- The deflation vectors are computed only once and can be reused throughout the simulation.
- If regions of constant permeability are contained and separated by large, well-defined jumps, then constructing a set of efficient deflation vectors automatically is relatively easy.

The default cases in this thesis do not have extreme discontinuities, but, as we will demonstrate, the permeability differences between (horizontal) layers are still large enough to benefit from physics-based deflation. Similar to the modification of the SPE5 case in the previous sections, we will also modify the SAGD permeability to demonstrate the potential of deflation.

5-4-1 Manual physics-based deflation

In the first set of numerical experiments with physics-based deflation, we manually assign deflation vectors. In particular,

- The BO case has two horizontal layers of constant permeability, as shown in Figure 5-1(c). To capture the jump, we manually construct two deflation vectors. In the first deflation vector, we assign value 1 to each cell in the top layer, and value 0 to each cell in the bottom layer. In the second deflation vectors, we assign value 0 to each cell in the top layer, and value 1 to each cell in the bottom layer. The boundary between the ones and zeros, in this case, coincides with the jump in permeability.
- The SPE5 case has three horizontal layers of permeability, as shown in Figure 5-1.3. The jump is captured using three deflation vectors, whose values are assigned in the same manner as above.
- The SAGD-SMALL case has a more complex permeability field. As discussed in Section 5-2-3, the eigenvectors corresponding to the isolated eigenvalues mainly

have non-zero values around the injector and the producer. Moreover, a closer look at Figure 5-1.7 reveals a number of horizontal layers of constant permeability in this region. Between these layers, rather large permeability jumps occur. Therefore, we assign ten deflation vectors to this region, where each vector represents a horizontal layer.

The deflation vectors for the BO, SPE5 and SAGD-SMALL case are illustrated in Figure 5-4.1(a), (b) and (c), respectively. In each figure, a front view is shown of the reservoir. Each color represents a deflation vector.

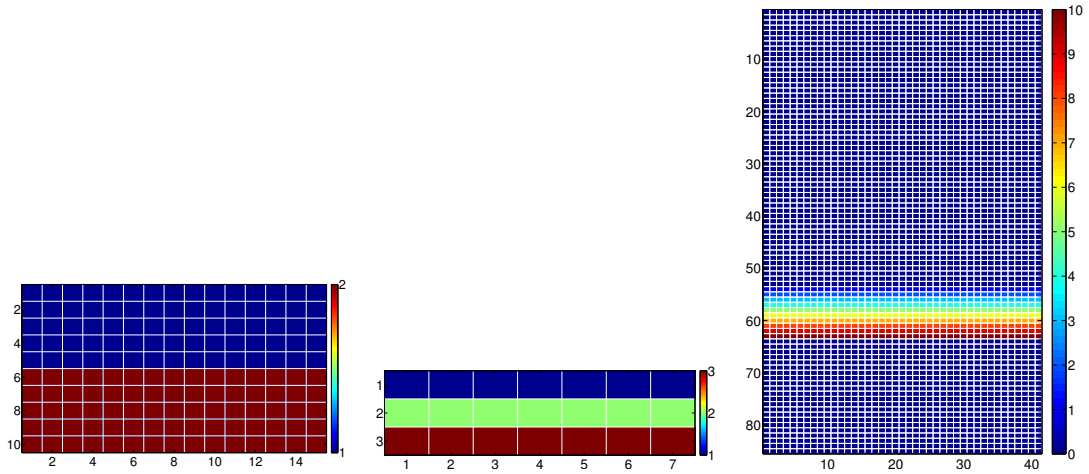


Figure 5-4.1: Manually constructed deflation vectors for (a) BO, (b) SPE5 and (c) SAGD-SMALL.

We now apply the manually constructed deflation vectors to the pressure solve. In each simulation, the pressure matrix is preconditioned with a Jacobi preconditioner, applied from the right. The residual tolerance is 10^{-6}, and, unless note otherwise, we use $m = 20$ with a maximum of 200 iterations. The convergence of physics-based deflation (PDGMRES) is compared to harmonic Ritz deflation (RDGMRES) and GMRES without deflation. In harmonic Ritz deflation, we use 2 deflation vectors for the BO case, 3 deflation vectors for the SPE5 case and 10 deflation vectors for the SAGD-SMALL case.

The convergence history of the BO case and the SAGD-SMALL case is shown in Figure 5-4.2.

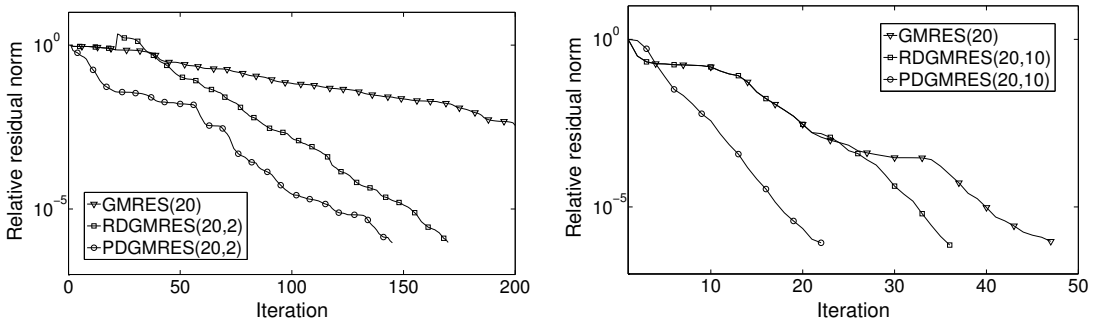


Figure 5-4.2: Comparison of no deflation, harmonic Ritz deflation (RDGMRES) and (manual) physics-based deflation (PDGMRES) for (a) BO and (b) SAGD-SMALL.

In Figure 5-4.2(a), GMRES(20) does not reach the tolerance within 200 iterations. The convergence of RDGMRES(20,2) is better, but, as observed before, a residual increase occurs after the restart. The best residual convergence is attained by physics-based deflation. After 20 iterations, the convergence speed of PDGMRES is approximately equal to the convergence speed of RDGMRES, however physics-based deflation can be applied from the start of the simulation which results in the best convergence. Similarly, PDGMRES in Figure 5-4.2(b) achieves the best performance. At the tolerance, the number of iterations compared to native GMRES is approximately halved. Again, the convergence speed of physics-based deflation after the restart is equal to the convergence speed of harmonic Ritz deflation, but the former method has the advantage of deflating the isolated eigenvalues from the start.

In figure, 5-4.3, we plot the convergence of default SPE5 solve and the solve of the SPE5 case with modified permeability ($\sigma = 10^6$).

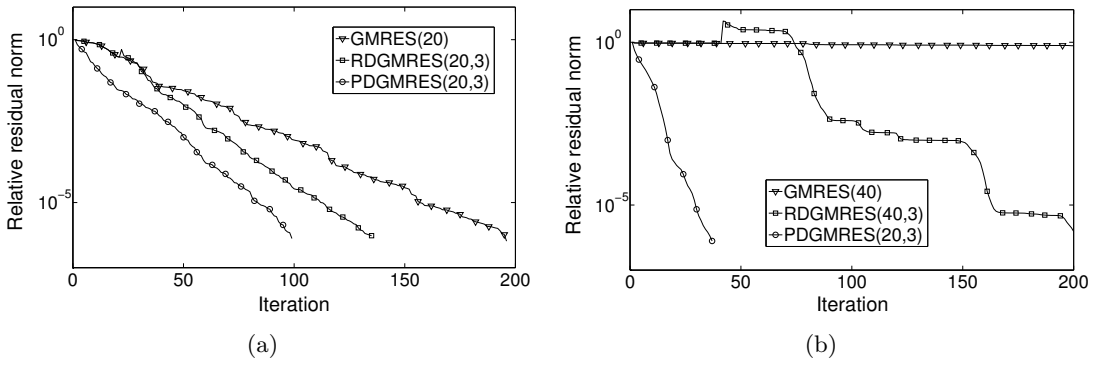


Figure 5-4.3: Comparison of no deflation, harmonic Ritz deflation (RDGMRES) and (manual) physics-based deflation (PDGMRES) for (a) SPE5 and (b) SPE5 with modified permeability ($\sigma = 10^6$).

The results in Figure 5-4.3(a) resemble the previously described phenomena. Physics-based deflation with 3 deflation vectors reduces the iteration count by about 100 iterations compared to GMRES and 40 iterations compared to harmonic Ritz deflation.

In the SPE5 case with modified permeability and $\sigma = 10^6$, the jump between the horizontal layers is relatively large. In line with the discussion of Figure 5-3.5, m is increased to 40 in Figure 5-4.3(b) to allow the harmonic Ritz vectors to converge to the eigenvectors corresponding to the extreme eigenvalues. RDGMRES reaches the tolerance after approximately 200 iterations. The convergence of PDGMRES is remarkably fast. This example illustrates that the span of the three manually constructed deflation vectors is a good approximation of the span of the eigenvectors corresponding to the extreme eigenvalues. Moreover, observe that we use $m = 20$ instead of $m = 40$. For $m = 20$, the convergence would be even better. We conclude that PDGMRES, compared to RDGMRES, not only achieves faster convergence but also does not impose any requirements on m, which, if m can be lowered, decreases the computational cost of GMRES.

In the last experiment of this section, we investigate the optimal maximum number of inner linear iterations, denoted M, in IX. By default, IX uses a single iteration to accelerate AMG in the pressure solve. To compete with this setup in terms of computational costs, the maximum number of iterations will have to be relatively low. The fact that physics-based deflation improves convergence from the start is especially advantageous (compared to harmonic Ritz deflation) for lower values of M. Table 5-4.1 compares GMRES without deflation to deflated GMRES with the 10 physics-based deflation vectors from Figure 5-4.1(c) in the SAGD-SMALL case. The default pressure solve tolerance of 10^{-2} is used.

	Non-linears		Fails		Outer linears		Inner linears	
M	PDGMRES	GMRES	PDGMRES	GMRES	PDGMRES	GMRES	PDGMRES	GMRES
1	278	272	1	1	1,347	1,568	1,347	1,568
2	268	279	0	1	1,135	1,1451	2,263	2,893
3	285	279	0	0	1,169	1,383	3,484	4,119
5	291	287	0	0	1,082	1,297	5,321	6,394
10	278	277	1	1	967	1,221	8,390	11,623
20	956	974	0	0	956	974	9,828	15,721

Table 5-4.1: PDGMRES and GMRES in the SAGD-SMALL case for varying M.

As the maximum number of inner iterations is increased, the amount of outer linear iterations decreases and the amount of inner linear iterations increases. Furthermore, if the pressure solve is allowed more iterations, then in general the residual will be smaller. As a result, the number of outer linears decreases. The number of non-linears stays approximately the same. Observe that the relative improvement of deflated GMRES over GMRES without deflation becomes more significant for higher values of M. For $M = 1$, the number of innear linears is decreased by 15%, compared to 37% for $M = 20$. This can be explained by the fact that for $M = 10$, and especially for $M = 20$, PDGMRES often converges to the tolerance before reaching the maximum number of iterations, whereas

GMRES does not. The tolerance is not reached for lower values of M, although the lack of failed non-linear iterations (or a large increase in non-linear iterations) indicates that the pressure residual is accurate enough after M iterations.

We aim to choose M such that PDGMRES has a significant advantage over GMRES, while limiting the computational costs of the pressure solve. Although the advantage of deflation is more significant for $M = 10$ or $M = 20$, the additional pressure iterations increase the cost. In our experience, $1 \leq M \leq 5$ is sufficient to prevent failed non-linear iterations, while the number of inner linear iterations is kept relatively low. In most of the experiments in the upcoming sections, we use $M = 2$. For this choice, as shown in the table above, deflation reduces both the inner linears and the outer linears by 22%.

5-4-2 Automatic physics-based deflation

As demonstrated by the Matlab experiments in the previous section, physics-based deflation has a number of advantages over harmonic Ritz deflation. The deflation vectors only have to be constructed once, and can be applied from the start of the simulation. As long as the span of the deflation vectors approximates the span of the eigenvectors corresponding to the extreme eigenvalues, physics-based deflation is both faster and cheaper. The example in Figure 5-4.3(b) shows that finding a suitable set of deflation vectors is relatively straightforward if the permeability field exhibits a number of distinct layers (or regions) of permeability, separated by large jumps.

The problem with manually assigning deflation vectors is that this process can become rather time-consuming and complex for large, realistic reservoirs. It is primarily for this reason that we have developed the partitioner algorithm from Section 4-2-1. The algorithm can handle any 2D or 3D permeability distribution. The main goal of the partitioner is to identify regions of constant permeability, and assign deflation vectors such that the jumps between the different regions correspond to the boundaries of the deflation vectors. We merge region-pairs with the smallest permeability differences until the desired number of deflation vectors is reached (or surpassed). The partitioner is implemented both in Matlab and IX.

In the first experiment, we export the SAGD-SMALL permeability from IX in order to run the partitioner in Matlab. We apply the partitioner to the same focus region around the injector and the producer as we use for the 10 manual deflation vectors in Figure 5-4.1(c). We aim to find a similar set of 10 deflation vectors using the partitioner, since we have visually confirmed that the 10 manual deflation vectors capture the jumps relatively well. The result is shown in Figure 5-4.4(a), next to the SAGD-SMALL permeability field.

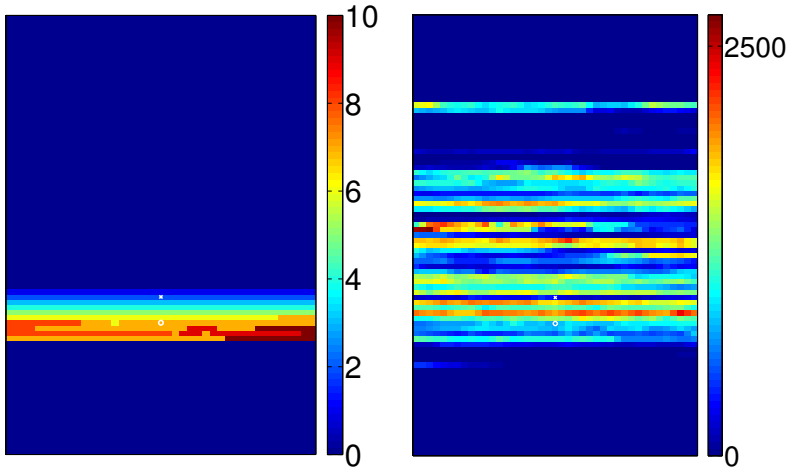


Figure 5-4.4: (a) Deflation vectors in the focus region and (b) SAGD-SMALL permeability.

The top five horizontal layers of the focus region are each assigned a deflation vector, similar to our manual construction in Figure 5-4.1(c). As confirmed by Figure 5-4.4(b), the boundaries of the deflation vectors coincide with several large permeability jumps. In the bottom five layers, the result from the partitioner and our manual construction is slightly different. Comparing the permeability in this region with the deflation vectors from the partitioner, we see that the permeability differences are more accurately captured than in our manual construction.

The 10 deflation vectors from the partitioner are compared to the 10 manual deflation vectors in Figure 5-4.5.

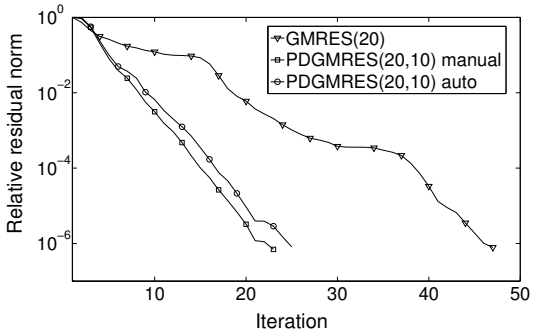


Figure 5-4.5: Comparison of physics-based deflation (PDGMRES) using manual or automatic construction of the deflation vectors.

The convergence of deflated GMRES using the manual set of deflation vectors (PDGMRES manual) is slightly better than the convergence of deflated GMRES using the 10

deflation vectors from the partitioner (PDGMRES auto). Apparently, capturing smaller details (i.e. jumps) does not necessarily improve convergence. Regardless, the difference in iterations is negligible and both methods are about twice as fast as normal GMRES.

As discussed in Section 4-3, the partitioner is particularly suitable for a parallel implementation. Each processor runs the partitioner on the corresponding parallel subdomain, which renders the parallel partitioner 'embarrassingly parallel'. Each resulting deflation vector has non-zero values only on the parallel subdomain on which it was constructed. Therefore, the deflation operators P_1 and P_2 can (mostly) be applied in parallel as well, as outlined in the second half of Section 4-3.

There is one possible issue with running the partitioner in parallel. If a permeability jump stretches across the boundary of two parallel subdomains, then the partitioner will assign two deflation vectors (one in each subdomain) instead of one deflation vector that captures the jump as a whole. As a result, an 'artificial' jump is introduced on the boundary of the parallel subdomain. In the application of deflation to structural mechanics by Jönsthövel in [13], this problem decreases the performance of deflated GMRES. We have conducted a number of experiments to investigate the impact of artificial jumps, and conclude that the problem does not affect convergence in our applications. In Figure 5-4.6, each deflation vector from Figure 5-4.1(c) is split vertically in half, which results in 20 deflation vectors.

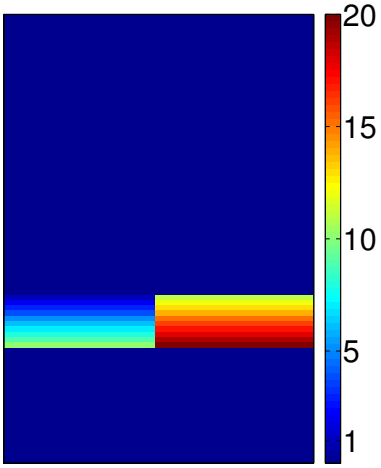


Figure 5-4.6: 10 manual deflation vectors split in 2.

The resulting convergence is shown in Figure 5-4.7 as PDGMRES(20,20). PDGMRES(20,40) uses 40 deflation vectors obtained by (vertically) splitting the 10 deflation vectors from Figure 5-4.1(c) in 4.

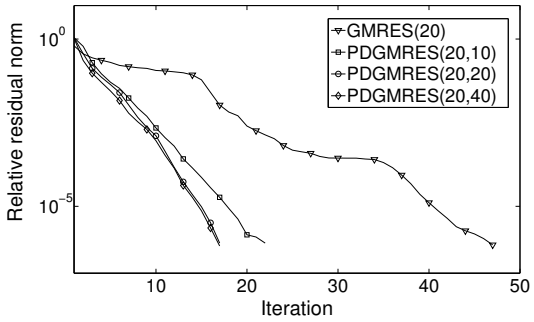


Figure 5-4.7: Comparison of deflation with 10 manual deflation vectors (PDGMRES(20,10)), 10 manual deflation vectors split in 2 (PDGMRES(20,20)) and 10 manual deflation vectors split in 4 (PDGMRES(20,40)).

Observe that deflation using the splitted deflation vectors is slightly faster than PDGMRES(20,10). Splitting the 10 deflation vectors in 2 or in 4 does not make a difference. Overall, we did not encounter results in Matlab that showed that splitting the deflation vectors is detrimental to the convergence. Table 5-4.2 shows the results from the same experiment in IX. The number of maximum inner iterations is 2.

	GMRES(20)	PDGMRES(20,10) (original)	PDGMRES(20,20) (10 vectors split in 2)
Non-linears	279	268	266
Fails	0	0	1
Outer linears	1,451	1,085	1,071
Inner linears	2,893	2,164	2,135
CPU time	5.16	6.58	10.12

Table 5-4.2: Comparison of PDGMRES(20,10) (10 manual deflation vectors) and PDGMRES(20,20) (10 manual deflation vectors split in 2) in IX.

We do not use the parallel implementation of P_1 and P_2 in this experiment, which is why the CPU time of the PDGMRES(20,20) linear solve is larger than the CPU time of the PDGMRES(20,10) solve. The number of outer iterations using the 10 manually constructed deflation vectors is approximately 25% lower than the outer linears in GMRES without deflation. Splitting the 10 deflation vectors in 2 does not result in slower convergence.

To prototype the partitioner in Matlab, we export the parallel partitioning from IX and independently apply the partitioner to each parallel subdomain. In a parallel run of the partitioner, the pre-specified desired number of deflation vectors is a maximum: subdomains with little fluctuation in the permeability can be assigned less deflation vectors, and sometimes the merging process surpasses the desired number of deflation

vectors. The latter is subject of future research. We compute at most 10 deflation vectors per subdomain. The parallel partitioning for $n_p = 8$, along with the permeability in x direction, is shown in Figure 5-4.8.

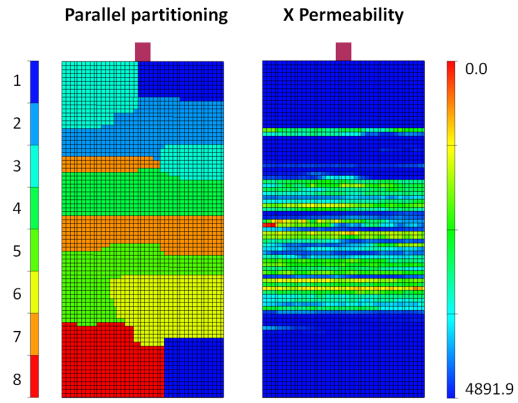
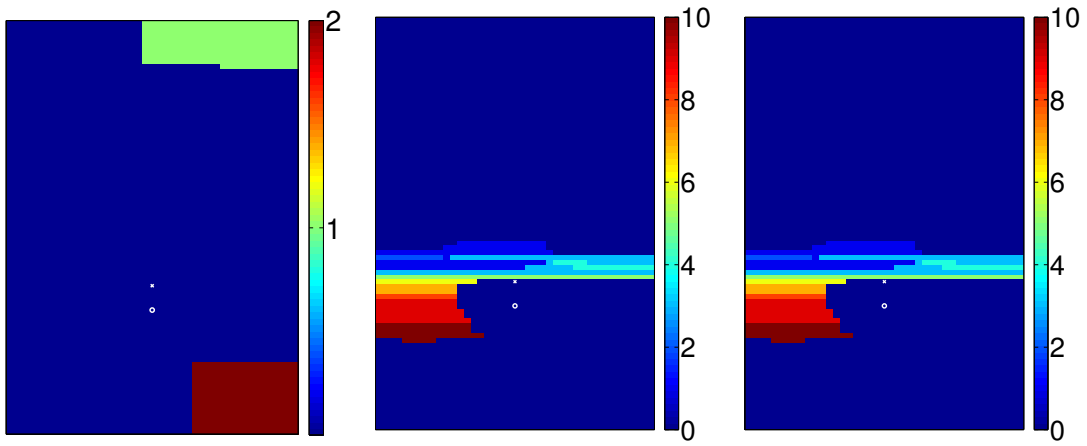


Figure 5-4.8: Parallel subdomains for $n_p = 8$ and the permeability in x direction.

The resulting deflation vectors for parallel domain 1, 5 and 6 are shown in Figure 5-4.9(a), (b) and (c), respectively.



(a) (b) (c)

Figure 5-4.9: Deflation vectors in the (a) first, (b) fifth and (c) sixth parallel subdomain.

We make several observations. Firstly, we see that the separated regions in the first parallel subdomain are assigned one deflation vector each. Indeed, no permeability jumps occur in this region. Note that this result illustrates why it is crucial to compute the maximum permeability (step two in the procedure outlined in 4-3) over all parallel subdomains. If we would have computed the local maximum, then the partitioner would

have assigned more deflation vectors to the different regions of permeability (not visible due to very small size). As these small differences do not give rise to extreme eigenvalues, the deflation vectors would be redundant. In future improvements of the partitioner, we suggest taking no deflation vectors in constant parallel subdomains, rather than assigning one deflation vector (as is currently the case).

Secondly, we observe that the horizontal layers in the focus region around the injector and the producer are captured. A close analysis of the deflation vectors in the fifth and sixth parallel subdomain does reveal that the algorithm is not flawless. On the left side of the fifth subdomain, some regions with a large permeability jump are merged together. Overall, however, the deflation vectors capture the permeability differences relatively well.

In figure 5-4.10, we vary the (maximum) number of deflation vectors per parallel subdomain in the SAGD-SMALL case with $n_p = 8$.

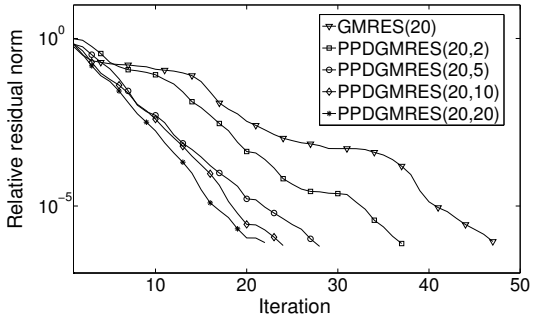


Figure 5-4.10: Comparison of parallel physics-based deflation (PPDGMRES) using eight processors and a varying maximum number of deflation vectors per parallel subdomain.

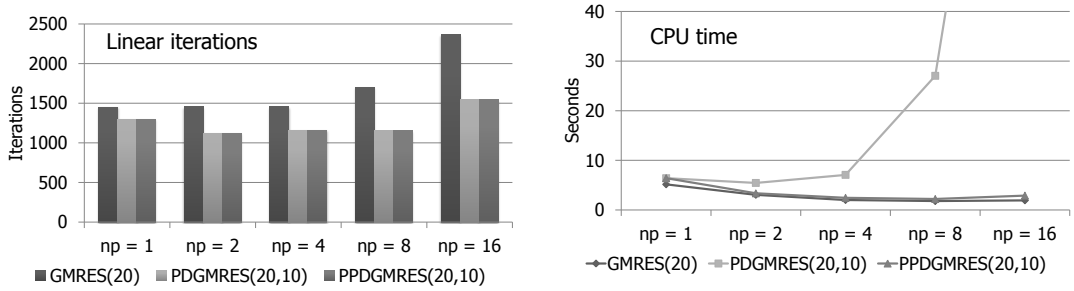
Notice that the decrease in iterations from PPDGMRES(20,2) to PPDGMRES(20,5) is much larger than the decrease from PPDGMRES(20,5) to PPDGMRES(20,10). Using only 2 deflation vectors, many permeability jumps are merged together and ignored. 5 deflation vectors, on the other hand, more accurately capture the large permeability differences. The convergence of PPDGMRES(20,10) and PPDGMRES(20,20) is similar, which implies that most of the jumps responsible for the harmful eigenvalues can be captured with 10 deflation vectors.

In the upcoming experiments, we test the parallel partitioner and the parallel implementation of deflation in IX. We vary the number of processors between 1 and 32, and use the SAGD-SMALL, SAGD-MEDIUM and SAGD-LARGE cases to experiment with different problem sizes. To compare serial deflation, parallel deflation and GMRES without deflation, we focus on the amount of inner linear iterations, and the CPU time of the linear solve. The maximum number of iterations is chosen as 2.

Note that we refer to parallel deflation and serial deflation as applying P_1 and P_2 in parallel (as outlined in 4-3) and in serial, respectively. This should not be confused with

the number of processors that is used. For example, with $n_p = 8$ and 10 deflation vectors per parallel subdomain (from the subdomain-levelset method), using serial deflation requires computations with $8 \times 10 = 80$ deflation vectors. If parallel deflation is used instead, the fact that each deflation vector is zero outside the corresponding subdomain theoretically reduces the computations to running deflation with only 10 deflation vectors. Some communication and global computations are required, so in practice parallel deflation is slightly slower.

Figure A-1.1 in Appendix A-1 shows the result of the parallel partitioner with a maximum of 10 deflation vectors per parallel subdomain. We verify that the deflation vectors are equal to the partitioner result in Matlab (Figure 5-4.9). Figure 5-4.11 illustrates the advantage of parallel physics-based deflation (PPDGMRES) over serial physics-based deflation (PDGMRES).

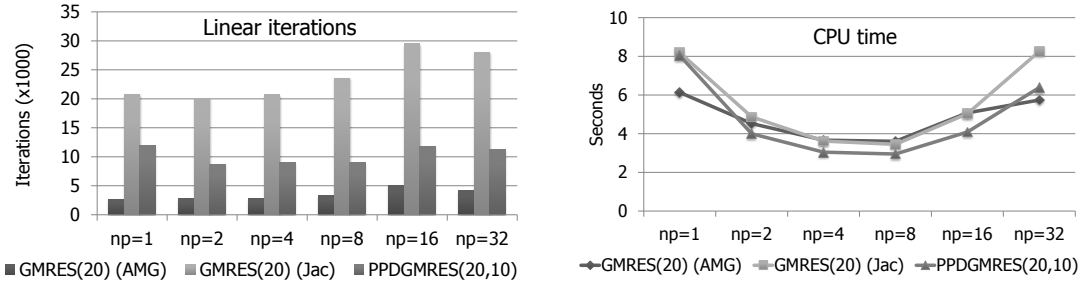


(a) (b)

Figure 5-4.11: Comparison of serial physics-based deflation (PDGMRES) and parallel physics-based deflation (PPDGMRES) in terms of (a) outer linear iterations and (b) CPU time of the linear solve in the SAGD-SMALL case (max. iterations = 2).

In Figure (a), we observe that the serial and parallel implementation of physics-based deflation results in the same amount of (outer) linear iterations. The relatively reduction in iterations increases from 11% for $n_p = 1$ to 35% for $n_p = 16$. This suggests that the quality of the deflation vectors becomes better, as the parallel subdomains become smaller. For serial deflation, the amount of deflation vectors in Z doubles as we double the number of processors. Twice as many deflation vectors results in four times as many elements in the Galerkin matrix E. The computational costs of the computations with Z and E increase quadratically, which explains the CPU time of PDGMRES in Figure (b). For parallel deflation, we observe that the overhead is minimal.

For a small maximum number of inner iterations, deflation is given little time to distinguish itself from GMRES without deflation. To highlight the potential of PPDGMRES, we temporarily set the maximum number of inner iterations to 40. The result is shown in Figure 5-4.12.



(a) (b)

Figure 5-4.12: Comparison of serial physics-based deflation (PDGMRES) and parallel physics-based deflation (PPDGMRES) in terms of (a) outer linear iterations and (b) CPU time of the linear solve in the SAGD-SMALL case (max. iterations $= 40$).

In Figure (a), we see that AMG requires the least amount of inner iterations. In most solves, AMG-preconditioned GMRES will reach the default tolerance within a few iterations. Jacobi-preconditioned GMRES is slower. By adding deflation, we observe that the number of iterations is approximately halved. For higher values of n_p, the advantage becomes even more apparent. The CPU times in Figure (b) show that PPDGMRES is the fastest method for $n_p = 2$, $n_p = 4$, $n_p = 8$ and $n_p = 16$. Note that the CPU times for Jacobi-preconditioned GMRES and AMG-preconditioned GMRES are similar for $n_p = 4$, $n_p = 8$ and $n_p = 16$. This can be explained by the fact that we are using a relatively small case, with a structured grid, for which a large number of (cheap) Jacobi iterations lead to the same CPU time as a small number of (expensive) AMG iterations.

In the next experiment, we use the SAGD-MEDIUM case. This case has 4 times as many rows in the pressure matrix as the SAGD-SMALL case. In addition, whereas SAGD-SMALL is a 2-dimensional case, SAGD-MEDIUM is 3-dimensional. In Figure 5-4.13, the permeability and deflation vectors (from the partitioner) on the sixth parallel subdomain are shown ($n_p = 8$). The rightmost figure shows the permeability in z direction restricted to one particular deflation vector.

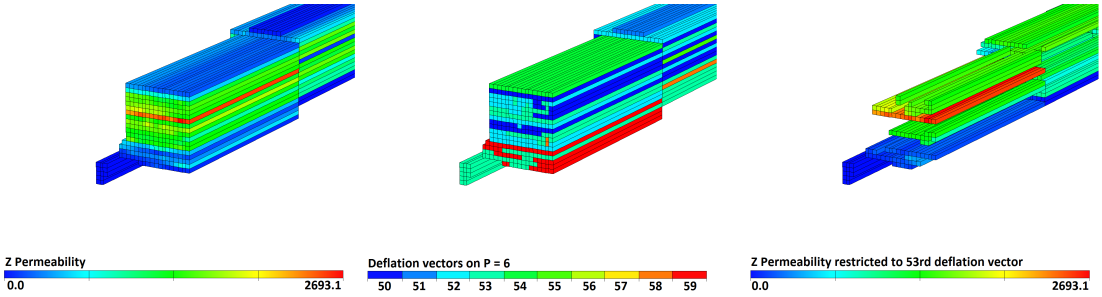
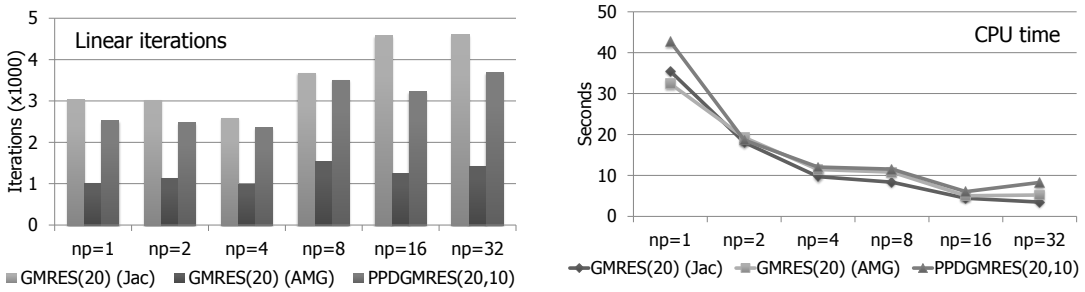


Figure 5-4.13: The permeability, deflation vectors and permeability projected on the 53rd deflation vector.

In the middle figure, we see that some jumps are accurately captured. The top horizontal layer of permeability shown on the left is captured in a single deflation vector, as shown by the green cells in the middle figure. Other layers, however, such as the (red) permeability layer in the center of the reservoir, are not appropriately captured in a deflation vector. The permeability is projected on the 53rd deflation vector, as shown on the right, which reveals the fact that both high and low permeability values are contained in this deflation vector. We expect this deflation vector to be ineffective. In addition, we expect that increasing n_p might remedy this problem, since the parallel subdomains for $n_p = 8$ are apparently too large to capture all permeability differences with 10 deflation vectors.

The convergence results of the SAGD-MEDIUM case for varying values of n_p is shown in the next figure.



(a) (b)

Figure 5-4.14: Comparison of serial physics-based deflation (PDGMRES) and parallel physics-based deflation (PPDGMRES) in terms of (a) outer linear iterations and (b) CPU time of the linear solve in the SAGD-MEDIUM case (max. iterations = 2).

As shown in Figure 5-4.14(a), up to $n_p = 8$ the reduction in iterations of PPDGMRES compared to GMRES (Jacobi) is minimal. Evidently, the SAGD-MEDIUM permeability field is more complex, and smaller parallel subdomains are required to capture the most

significant permeability jumps and obtain good approximations of the eigenvectors. Indeed, for $n_p = 16$, the set of deflation vectors is more effective, as the number of linear iterations is reduced by 30%. For $n_p = 32$, however, the reduction is only 20%. As shown in Figure (b), the CPU time of the linear solve of PPDGMRES is comparable or slightly higher than GMRES.

In the next example, we apply the partitioner and parallel physics-based deflation to the largest case in this study: SAGD-LARGE.

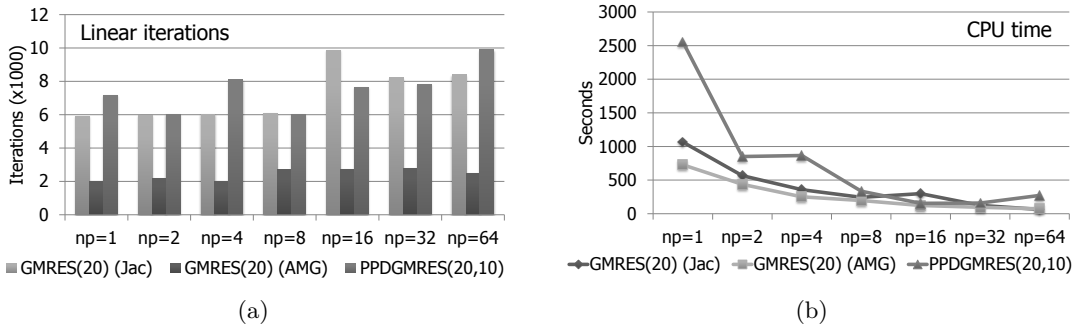


Figure 5-4.15: Comparison of serial physics-based deflation (PDGMRES) and parallel physics-based deflation (PPDGMRES) in terms of (a) outer linear iterations and (b) CPU time of the linear solve in the SAGD-LARGE case (max. iterations = 2).

We observe that deflation is not robust anymore. For $n_p = 16$, PPDGMRES reduces the inner linear iterations, but for all other values of n_p the iterations are similar or higher. Consequently, deflation does not improve the CPU time of the linear solve, as shown in Figure 5-4.15. We conclude that the partitioner produces effective deflation vectors for the SAGD-SMALL case, but for the larger permeability fields in SAGD-MEDIUM and SAGD-LARGE the automatically constructed physics-based deflation vectors do not improve convergence.

In the last numerical experiments of this section, we modify the permeability of the SAGD-MEDIUM case. We suspect that the permeability in the SAGD-MEDIUM and SAGD-LARGE cases does no produce (enough) isolated and/or extreme eigenvalues for deflation to be efficient. In order to investigate if the partitioner and parallel physics-based deflation can also be effective for larger cases, we modify the SAGD-MEDIUM permeability field such that more jumps are introduced. A front view of the 'inflated' permeability distribution in x direction is shown in Figure 5-4.16.

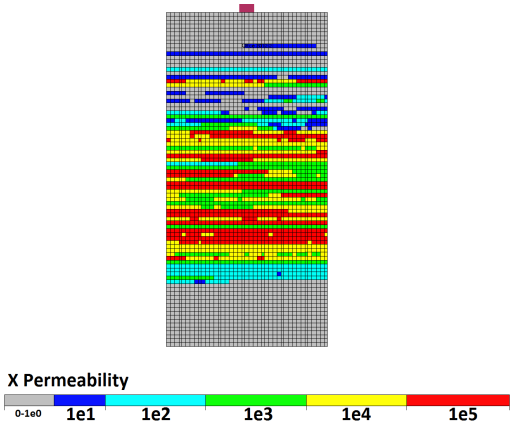


Figure 5-4.16: Inflated permeability field of the SAGD-MEDIUM case.

The permeability values in the x and y direction above 4000, for example, are inflated to 10^5. Other ranges of permeability are similarly inflated (or in some cases, deflated) to create large permeability differences. Due to the size of the SAGD-MEDIUM case, we cannot plot the spectrum in Matlab. We can, however, inflate the SAGD-SMALL case (using the same procedure) and plot the spectrum, to get an indication of the occurrence of small eigenvalues. Note that the front vertical permeability layer of the SAGD-MEDIUM case is equal to the permeability in the SAGD-SMALL case. Therefore, the inflation will lead to the same field as shown in Figure 5-4.16. The spectrum of the inflated SAGD-SMALL case is shown in the next figure.

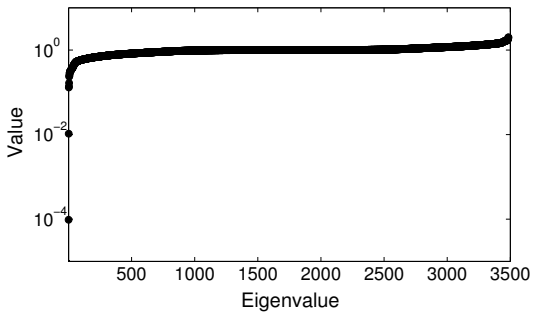
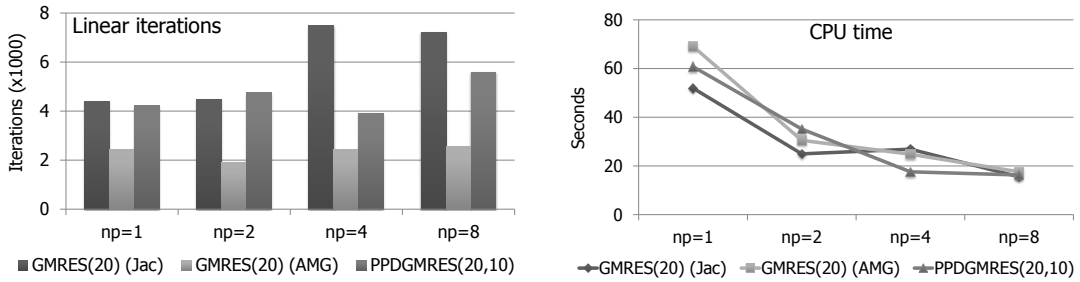


Figure 5-4.17: Spectrum of the first inflated SAGD-SMALL case.

Whereas in Figure 5-2.10 and Figure 5-2.11 we do not encounter any isolated eigenvalues smaller than 10^{-3}, we now find an extreme eigenvalue of size 10^{-4}. Given the fact that there are many large permeability jumps in 5-4.16, the correspondence between the number of isolated eigenvalues (only 2 in this case) to the number of permeability jumps (observed in Section 5-2-2) clearly does not hold.

Regardless, as shown in the results in Figure 5-4.18, the performance of deflation improves compared to the default SAGD-MEDIUM results in Figure 5-4.14.



(a) (b)

Figure 5-4.18: Comparison of serial physics-based deflation (PDGMRES) and parallel physics-based deflation (PPDGMRES) in terms of (a) outer linear iterations and (b) CPU time of the linear solve in the first inflated SAGD-MEDIUM case (max. iterations = 2).

For $n_p = 4$, the number of linear iterations compared to Jacobi-preconditioned GMRES is reduced by almost 50%. The CPU time for PPDGMRES is lower than both Jacobi- and AMG-preconditioned GMRES. This result does not extend to $n_p = 8$. Here, the reduction is only 23%, and the CPU times of all methods are approximately equal. We believe that the reason for this discrepancy is that the deflation vectors constructed by the partitioner for 4 parallel subdomains happen to be more effective in capturing the jumps than the deflation vectors for 8 parallel subdomains, even though the parallel subdomains are smaller. In Figure 5-4.13 we have seen that the opposite can also be true: the deflation vectors were more efficient for $n_p = 16$ than for $n_p = 8$.

In Figure 5-4.19, another modification of the SAGD-MEDIUM permeability field is shown.

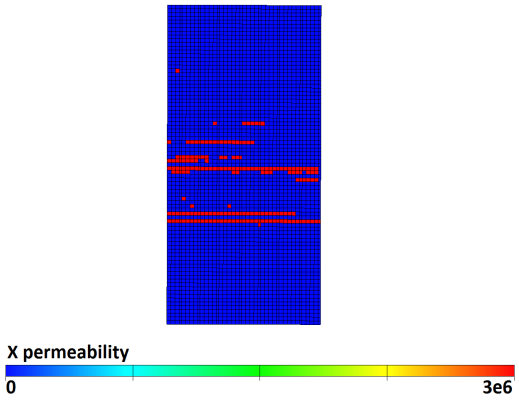
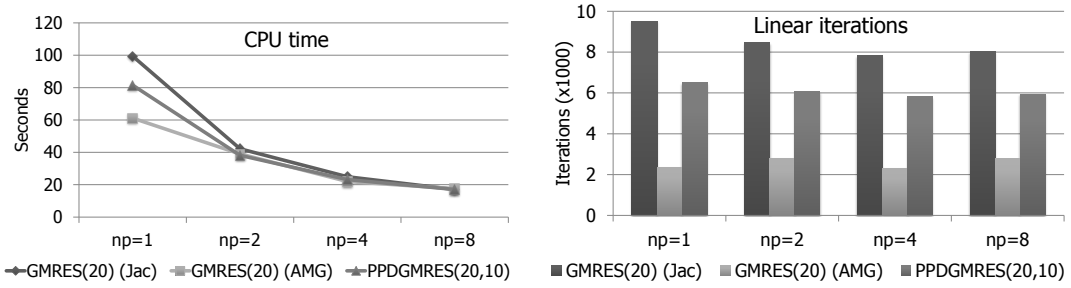


Figure 5-4.19: Spectrum of the second inflated SAGD-SMALL case.

This time, we aim to imitate fractures. We expect that this permeability field will result in more extreme eigenvalues than the modified field in 5-4.16. Indeed, the convergence

shown in Figure 5-4.20 is decent for all values of n_p.



(a) (b)

Figure 5-4.20: Comparison of serial physics-based deflation (PDGMRES) and parallel physics-based deflation (PPDGMRES) in terms of (a) outer linear iterations and (b) CPU time of the linear solve in the second inflated SAGD-MEDIUM case (max. iterations = 2).

We conclude that the partitioner is not robust in the large 3-dimensional cases. Especially in the mix of homogeneous and heterogeneous regions of permeability in the default SAGD-MEDIUM and SAGD-LARGE cases, the partitioner does not succeed in constructing a good set of deflation vectors. Increasing the number of processors, as we reasoned could lead to better results, does not work in this case. We do see, however, that for the first, and especially the second modified SAGD-MEDIUM field (resembling fractures), the convergence is better. This indicates that although there is room for improvement in the partitioner, we also find that the default cases we have been using are not the most suitable for deflation due to a lack of large permeability jumps and extreme eigenvalues.

5-4-3 Summary of findings

In this section, we demonstrate the potential of using parallel physics-based deflation. The subdomain-levelset method gives rise to a set of deflation vectors that are particularly suitable for fast parallel computations. Initially, we manually assign deflation vectors based on a visual analysis of the permeability field. In our Matlab experiments, we find that deflation can significantly improve convergence, mainly because physics-based deflation works from the start of the simulation. For very large permeability jumps, such as in the SPE5 case with modified permeability and $\sigma = 10^6$, the advantage of physics-based deflation over harmonic Ritz deflation is remarkable. We investigate the optimal maximum number of inner linear iterations, and, based on a consideration of the effectiveness of deflation and the computational costs, we conclude that 2 iterations is a good choice.

In the experiments with the partitioner, we demonstrate that the resulting deflation vectors capture the permeability jumps in the SAGD-SMALL case. As a result, we achieve good speedups both in Matlab and IX. We also show that there is no negative impact of introducing artificial jumps, i.e. separated deflation vectors that span the same (often horizontal) jump in permeability. Lastly, in our experiments with the SAGD-MEDIUM and SAGD-LARGE case, we find that parallel physics-based deflation is not robust anymore. We believe that this is due to the fact that the partitioner is not able to capture the permeability jumps in the default, complex 3-dimensional permeability field. Experiments with a modified permeability field in the SAGD-MEDIUM case highlight the fact that the partitioner works better for cases with stronger heterogeneity.

5-5 Other strategies

In addition to harmonic Ritz deflation and physics-based deflation, several other methods are tested. In this section, we highlight two initiatives: using the saturation or mobility instead of the permeability to construct the deflation vectors, and applying deflation to the reservoir system to remedy the loss of information in the parallel ILU decomposition.

5-5-1 Deflation using saturation or mobility

As shown for the SPE5 case in Figure 5-2.4 and for the SAGD-SMALL case in Figure 5-2.12, the eigenvectors corresponding to the smallest eigenvalues contain patterns that match the flow in the reservoir. This observation gives rise to the idea of using the saturation, instead of the permeability, to construct the deflation vectors. Recall from Equation (2-1.1) that the PDE coefficient in the governing equations of a two-phase flow is the mobility. From Equation (2-1.3), we know that the mobility is computed by multiplying the viscosity, the absolute permeability (which we have used in deflation so far) and the relative permeability. The relative permeability is directly determined by the saturation. We always assume that the absolute (or initial) permeability is the dominant coefficient in the mobility, but the observations in Figure 5-2.4 and Figure 5-2.12 suggest that the saturation may also play a role.

In the SPE5 case, two deflation vectors are used. The first deflation vectors contains ones in cells with a saturation higher than a certain threshold and zeros elsewhere. The second deflation vector is the complement. As a result, the jump between the flooded and unflooded regions is captured. We test this idea in Matlab, and find that the convergence of deflated GMRES does not improve. Even though the eigenvectors of the SPE5 case in Figure 5-2.4 do not show the permeability layers until the reservoir has reached an equilibrium, we find that the deflation vectors based on the three layers of initial permeability are the most efficient for all time steps. We suspect that the saturation jump between the flooded and unflooded regions is too small to produce isolated

or extreme eigenvalues. Regardless, using the saturation in physics-based deflation requires recomputing the deflation vectors at every timestep. This render the method too expensive in practice.

We also test the mobility in the partitioner. We find that the results are comparable to using the permeability to construct the deflation vectors. This suggests that the absolute permeability is the dominant coefficient in the mobility, while the influence of the saturation is limited. Since the mobility changes at every time step (due to the change in relative permeability), we prefer the absolute permeability (which is constant throughout the simulation) in our physics-based deflation experiments.

5-5-2 ILU damage

In Figure 5-2.17, we see that non-zero values appear on the parallel subdomain boundaries in the eigenvectors of the reservoir matrix. These values occur because connections between subdomains are ignored in the blockwise ILU decomposition. We test two approaches:

1. n_p deflation vectors, where each deflation vector holds ones in the cells on a parallel subdomain and zero elsewhere.
2. Deflation vectors with ones on the boundaries of the parallel subdomains

The first approach is essentially subdomain deflation. By assigning 1 deflation vector per parallel subdomain, we hope to recover information that is lost in the incomplete LU decomposition. In the second approach, we aim to approximate the eigenvectors from Figure 5-2.17. We also test the second approach combined with smoothed aggregation (see e.g. [13]) to 'push' the deflation vector towards the real eigenvector.

None of the methods lead to improved convergence. Our focus is on harmonic Ritz deflation and physics-based deflation, hence we do not further investigate this deflation approach.

Chapter 6

Conclusions

The focus of this thesis is on deflation methods in reservoir simulation. We conduct numerical experiments with harmonic Ritz deflation and physics-based deflation, using a range of test cases with varying dimensions and degrees of complexity.

6-1 Summary of theory

In the theoretical sections of this thesis, we introduce a mathematical framework for the IX engine and deflation. We start with a discussion of the reservoir equations for two-phase flow. The IX formulation requires a number of additional assumptions and components, giving rise to a coupled linear system of reservoir and well equations. The non-linear equations are solved using the Newton-Raphson method. We discuss the time step size selection that is important for the Adaptive Implicit Method (AIM), although in our experiments, only a fully implicit model is used.

The linear solver in IX, GMRES, is introduced in a discussion of Krylov subspace methods. We prove that the Arnoldi method can be used to generate an orthonormal basis of the Krylov subspace, and discuss the Ritz values and Ritz vectors produced in this process. Using the Givens rotations and a QR-factorization, the least-squares problem in GMRES can be effectively solved. We also show that the Givens rotations can be exploited to find the residual norm. We provide an overview of the convergence theory for GMRES, and conclude that the residual bounds for non-symmetric matrices provide, at most, an indication of the residual reduction.

Preconditioning a Krylov subspace method can significantly improve convergence and robustness. We state the pseudocode for left- and right-preconditioning, and attribute

our preference for right-preconditioning to the ease with which the residual norm can be computed. The Constrained Pressure Residual (CPR) preconditioner used in IX is discussed in detail. We highlight the advantages of the method, namely the ability to individually tackle the elliptic nature of the pressure equations and the hyperbolic nature of the remaining equations. The former are resolved with an Algebraic Multigrid (AMG) preconditioner, and for the latter, an incomplete LU factorization is used. The advantage of both AMG and ILU is that these methods are robust for a broad range of problems, while the shared disadvantage is the lack of scalability.

We motivate the use of deflation using the Ritz values. We show that 'natural' deflation occurs after the Ritz values have converged to the smallest eigenvalues in the spectrum, which gives rise to 'artificial' deflation. We aim to deflate isolated and extreme eigenvalues, which harm the convergence of GMRES, by approximating the corresponding eigenvectors. Different deflation methods are discussed, from which we conclude that static deflation methods using a projection preconditioner are the most suitable for our applications. In the corresponding mathematical framework, we present relevant deflation theory on the formulation of the preconditioners and the convergence. Furthermore, we compare a number of methods to compute the deflation vectors, including exact eigenvectors, (harmonic) Ritz vectors and domain-based vectors. We find that the latter two are the most promising, because of the black-box nature of harmonic Ritz deflation and the ability of domain-based deflation to capture permeability jumps. The permeability jumps are responsible for the occurence of extreme eigenvalues, which, in turn, harm the convergence of GMRES.

The implementation of harmonic Ritz deflation and subdomain-levelset deflation, a particular domain-based approach, is introduced in terms of the pseudocode. The levelset method, in which deflation vectors are assigned to constant regions of permeability separated by large jumps, initiates the development of the partitioner algorithm. Ideally, the boundaries of the deflation vectors overlap with the permeability jumps. We introduce the partitioner through a step-by-step treatment of a small example, and refer to the appendix for the full pseudocode. The partitioner is particularly suitable for a parallel implementation. We describe the 'embarrassingly parallel' nature of the algorithm, and also show that, because of the sparsity, the resulting deflation vectors can be used in an efficient parallel implementation of the deflation preconditioners. We highlight the fact that subdomain-levelset deflation could be more effective as we increase the number of processors, for reasons described in the next section.

6-2 Conclusions in the results

We describe the five cases used in this thesis in terms of the dimensions, the initial conditions, wells and performance in the Newton-Raphson method. We highlight the horizontal layer structure in the the permeability distribution of all our cases, as well as the complexity of the steam injection method. The spectra of all cases are analyzed.

In our cases, we find that the default permeability produces isolated, but not extreme, eigenvalues. Therefore, we modify the permeability in the SPE5 case to demonstrate the correspondence between the number and size of the extreme eigenvalues, and the number and size of the permeability jumps. As expected, it is shown that the size of the extreme eigenvalues negatively influences the convergence of GMRES. Analyzing the eigenvectors of the steam injection case reveals that non-zero values occur mainly around the injector and producer. Lastly, we use the spectrum to verify that AMG is a more effective preconditioner than diagonal scaling for the pressure solve in IX.

Numerical experiments with exact eigenvectors as deflation vectors demonstrate the potential of deflation. The performance improvement is particularly evident for the modified SPE5 case (with very large jumps in the permeability). Harmonic Ritz deflation is subsequently shown to be effective, as long as the cycle size is sufficiently high to allow the harmonic Ritz vectors to converge to the exact eigenvectors. Furthermore, we show that the preconditioner can be left out in the formulation of the deflation operators, which is crucial for the implementation in IX. We compare appending deflation vectors and freezing deflation vectors, and conclude that the increased computational costs of appending do not outweigh the benefit in convergence.

In the IX experiments with harmonic Ritz deflation, we compare the results mainly based on the number of pressure iterations and the CPU time of the full linear solve. Using deflation in the SPE5 case, the number of pressure iterations is decreased enough to outweigh the overhead of deflation. For other cases, however, these results do not hold and deflation is too expensive to improve the performance of the pressure solve. Furthermore, AMG-preconditioned GMRES without deflation is significantly faster than Jacobi-preconditioned GMRES combined with harmonic Ritz deflation. Combining AMG with deflation does not improve the convergence, most likely due to the fact that AMG is already very effective in tackling the error corresponding to the extreme eigenvalues. Our experiences with harmonic Ritz deflation motivate the use of physics-based deflation, which is theoretically both cheaper and more effective.

In physics-based deflation, we either use manually constructed deflation vectors or the subdomain-levelset method to obtain deflation vectors that capture the heterogeneity in the underlying physics. Using a visual analysis of the permeability distributions, we first manually assign deflation vectors to the distinct layers of permeability. Used in a deflation method, the deflation vectors are shown to be effective in improving the performance. The convergence speed is approximately equal to harmonic Ritz deflation. Physics-based deflation, however, can be used from the start of the linear solve, rather than after the first cycle, which results in a larger reduction in iterations. The manual deflation vectors are particularly effective for cases with very large permeability jumps.

Manually constructing deflation vectors in large, realistic reservoirs can be a time-consuming and complex process. To this end, we employ the partitioner to automatically identify regions of constant permeability, and assign deflation vectors such that the jumps

between the different regions correspond to the boundaries of the deflation vectors. We confirm for the steam injection case that the deflation vectors produced by the partitioner approximately coincide with the manual set of deflation vectors. Convergence for both approaches is shown to be the same. In the parallel implementation of the partitioner, we identify a possible issue with introducing 'artificial' jumps. Although in some applications the artificial jumps are detrimental to the convergence, we find that separating permeability jumps in different deflation vectors of the steam injection case does not decrease the performance of the linear solve, both in IX and Matlab. Consequently, we reason that, theoretically, the effectiveness of the deflation vectors should improve as we increase the number of processors. For a fixed number of deflation vector per parallel subdomain, decreasing the size of the subdomains allows the deflation vectors to capture the permeability jumps in more detail.

Lastly, we test the partitioner and parallel deflation in IX. We find that the parallel implementation requires significantly less CPU time in the linear solve, compared to deflation applied in serial. For the 2-dimensional steam injection case, the physics-based deflation method is also effective in reducing the pressure iterations. For some runs, deflated Jacobi-preconditioned GMRES is faster than AMG-preconditioned GMRES without deflation, although the difference is minimal. As we test larger, realistic 3-dimensional cases, we find that the performance of the deflation vectors from the partitioner deteriorates. The cost of deflation increases, while the reduction in iterations is small or absent. Modifying the permeability to exhibit more heterogeneity shows that the dimensions are not per se the problem, since better convergence is achieved when more heterogeneity is introduced.

The final conclusion of this thesis is that the subdomain-levelset method and parallel deflation have potential to yield excellent improvements of the performance, but improvements are needed in the partitioner to construct robust deflation vectors for larger, more complex permeability fields. In addition, the method should be tested for cases with stronger heterogeneity.

6-3 Future research

We mainly recommend further research into the subdomain-levelset partitioner algorithm. In retrospect, we conclude that the algorithm is set up to work well for permeability distributions that do not necessarily produce extreme eigenvalues. The partitioning and merging process is effective for a mix of heterogeneous and homogeneous permeability regions (such as in the SAGD case), but an analysis of the code reveals that the partitioner is less effective if we have a few isolated areas that differ significantly from the rest of the permeability field. The latter could be the case, for example, in an underground river or a fracture. The SPE5 case with modified permeability illustrates that, potentially, deflation can significantly reduce the number of iterations for these cases.

We show in the results that the best convergence improvement is achieved when all extreme eigenvalues are deflated. Leaving even a single extreme eigenvalue in the spectrum, especially if the eigenvalue is very small, results in a significant increase in iterations. The first improvement, therefore, is to allow the partitioner to compute the number of deflation vectors per parallel subdomain. By doing so, we ensure that all permeability jumps are captured. This is particularly important for the cases with fractures or underground rivers described before. If a large number of significant permeability jumps exist in a relatively small region, then the partitioner might construct (too) many deflation vectors. In this case, we suggest increasing the number of processors, as this will decrease the size of the subdomains. Ideally, although we are unsure how this will affect other components of IX, we would include the number of large permeability jumps as a criterion in the parallel partitioning (by PARmetis), similar to the work in [34].

Our other suggestions for future research on the implementation are as follows:

- More research is needed on the splitting of deflation vectors, i.e. in the case that the boundary of a parallel subdomain crosses through a connected permeability jump. Our experiments with the SAGD-SMALL case indicate that the performance remains the same, but the results in [13] argue for the opposite. It is possible that introducing the artificial jumps in the SAGD-MEDIUM and SAGD-LARGE case is the cause of the lack of robustness. As a possible remedy, similar to [13], we propose to investigate the overlap of deflation vectors. The 'weighted overlap' approach in [62] might be particularly suitable for our applications.
- In the partitioner itself, several improvements can be made to increase the speed of the algorithm:
  - We suggest limiting the deflation vectors to regions where flow occurs. In the SAGD case, we find that non-zero values in the eigenvector appear only around the injector and producer. In addition, the 10 manually constructed deflation vectors from Figure 5-4.1(c) perform well. Therefore, computational costs could be reduced by forcing the partitioner to construct the deflation vectors in a focus region. The consequences of this approach for the load balancing should be examined.
  - As discussed in the results, currently one deflation vector is assigned to a parallel subdomain without jumps. We suggest assigning no deflation vectors to such regions, to decrease the computational costs.
- We suggest investigating the use of a second criterion, in addition to the summed permeability jump, in the merging process: the number of adjacent cells. For example, imagine the situation where we have two regions, both adjacent to a third region. The first region has a few cells with a large permeability jump to adjacent cells in the third region, and the second region has many cells with a small permeability jump to adjacent cells in the third region. If the summed permeability difference between the first region and the third region is smaller

than the summed permeability difference between the second region and the third region, then the first region will be merged before the second. Our results show that the first region is more likely to cause extreme eigenvalues, which is why this situation is undesirable. We suggest implementing a second region-adjacency matrix (in addition to the jump matrix) for the number of adjacent cells. The initial thresholds and threshold increments for both the permeability difference and number of adjacent cells will have to be tuned to yield the most desirable set of deflation vectors.

Lastly, we suggest further modifying the SPE5 permeability field (or a comparable case) to imitate small permeability patterns that we might encounter in reservoirs. In this report, only a 'sandwich' was tested, but we can also experiment with other structures. Plotting the eigenvalues and eigenvectors will provide insight in the desirable shape of our deflation vectors. We emphasize the fact that the convergence of deflated GMRES should be tested in cases for which deflation is potentially the most effective, e.g. underground rivers and fractures, instead of the relatively homogeneous permeability field of the SAGD cases.

Appendix A

Partitioner pseudocode

In this section, we discuss the pseudocode for the five functions of the partitioner (Figure 4-2.7). The notation for the partitioner pseudocode is summarized in Table A-0.1.

Notation	Meaning
A	$n_c \times n_c$ adjacency matrix
$\mathcal{A}_i$	Set of (the indexes of) the nodes adjacent to node i
β	Boolean operator to indicate if change occured
i, j, k	Either cell (partition) or region (merge) numbers
κ_i^x	The x coordinate of node i in the (Intersect) computational domain
κ_i^y	The y coordinate of node i in the (Intersect) computational domain
κ_i^z	The z coordinate of node i in the (Intersect) computational domain
$\mathcal{L}_1$, $\mathcal{L}_2$	Temporary lists of indices
$\mathcal{N}_p$	Set of all cell numbers, i.e. $\mathcal{N}_p = \{1, 2, \ldots, n_c\}$
$\mathcal{N}_r$	Set of all curent region numbers, i.e. $\mathcal{N}_r = \{1, 2, \ldots, n_c\}$
n_d	**Presecified** (maximum) number of regions after the merging
n_c	Number of cells (or points) in the domain
n_r	Intermediate number of regions in s
n_{ra}	**Prespecified** number permeability range categories
p	$n_c \times 3$ vector consisting of p_x, p_y and p_z
p_x	$n_c \times 1$ vector of permeability values in x-direction
p_y	$n_c \times 1$ vector of permeability values in y-direction
p_z	$n_c \times 1$ vector of permeability values in z-direction
p_{max}	Maximum of P
p_{range}	p_{max} divided by n_{ra}
s	$n_c \times 1$ vector of region numbers
W	$n_r \times n_r$ jump matrix
$\mathcal{W}_i$	Set of (the indexes of) the regions adjacent to region i
ω	**Prespecified** threshold for the permeability jump
ω_+	**Prespecified** value of the increment for the threshold

Table A-0.1: Partitioner pseudocode notation.

Remark A-0.1.

- *In practice, $\mathcal{A}_i$ and $\mathcal{W}_i$ are derived from row i of A and W, respectively.*
- *Vector elements are denoted with square brackets, e.g. $s[i]$, and matrix elements are written as $A[i][j]$.*
- *Indexing of vectors and matrices starts with* 1.

The parameters n_d, n_{ra}, ω and ω_+ have to be prespecified. In most of our experiments, we use $n_{ra} = 100$, $\omega_0 = 100$ and $\omega_+ = 100$. The variable n_d is varied in our numerical experiments to analyze the convergence of deflated GMRES using a varying number of deflation vectors.

The pseudocode for main, initialize, partition, merge and jumps is given in Algorithm 7, 8, 9, 10 and 11, respectively.

Algorithm 7 Subdomain-levelset partitioner: main

Input: $A \in \mathbb{R}^{n_c \times n_c}$ and $p \in \mathbb{R}^{n_c \times 3}$
Output: $Z \in \mathbb{R}^{n_c \times \hat{n}}$, for some $\hat{n} \leq n_d$
$p_{\max} = \max(p_z)$ and $p_{\text{range}} = p_{\max}/n_{ra}$
Call $s =$ initialize(A, p_z, p_{range})
Call $s =$ partition(A, s)
Call $s =$ merge(A, s, p, n_d)
$n_r = \max(s) + 1$
Initialize $Z \in \mathbb{R}^{n_c \times n_r}$
for $j = 1, 2, \ldots, n_r$ **do**
  for $i = 1, 2, \ldots, n_c - 1$ **do**
    if $s[i] = j$ **then**
      $Z[i][j] = 1$
    end if
  end for
end for

The main program receives the adjacency graph and the permeability values in $x-$, $y-$ and $z-$ direction, and constructs the deflation vectors using the levelset method. Note that p_{max} and p_{range} could be evaluated in initialize instead of main. We choose to do the computations in main for reasons related to the parallel implementation. Details on this topic will be given in Section 4-3.

Since the porous media can exhibit different permeability properties in different directions, we have in general $p_x \neq p_y \neq p_z$. The variable p_z is most important in our cases, as the permeability layers are stretched horizontally and jumps occur most often in the vertical direction. Hence, we have chosen to use p_z for the initial partitioning in initialize. In merge, all three directions are used to determine which regions should be taken together.

Algorithm 8 Subdomain-levelset partitioner: initialize

Input: $A \in \mathbb{R}^{n_c \times n_c}$, $p_z \in \mathbb{R}^{n_c \times 1}$ and $p_{\text{range}} \in \mathbb{R}$
Output: $s \in \mathbb{R}^{n_c \times 1}$
Initialize $s \in \mathbb{R}^{n_c \times 1}$
for $i = 1, 2, \ldots, n_c - 1$ **do**
  $k = 0$
  $\beta =$ false
  while $\beta =$ false **do**
    if $p_z[i] = 0$ **then**
      $s[i] = 0$
      $\beta =$ true
    else if $k \cdot p_{\text{range}} < p_z[i] \leq (k+1) \cdot p_{\text{range}}$ **then**
      $s[i] = k$
      $\beta =$ true
    end if
    $k = k + 1$
  end while
end for

The permeability in $z-$direction of each node is placed in one of the n_{ra} categories of permeability range. The scalar k is incremented until the correct category has been found.

Algorithm 9 Subdomain-levelset partitioner: partition

```
1: Input: A ∈ ℝ^{n_c×n_c} and s ∈ ℝ^{n_c×1}
2: Output: s ∈ ℝ^{n_c×1}
3: 𝒩_p = {1, 2, ..., n_c} and c = 0
4: s_temp = s
5: while 𝒩_p ≠ ∅ do
6:    Pick any i ∈ 𝒩_p
7:    𝒩_p = 𝒩_p\{i}
8:    ℒ_1 = {i}
9:    s[i] = c
10:   while ℒ_1 ≠ ∅ do
11:      ℒ_2 = ∅
12:      for all j ∈ ℒ_1 do
13:         for all k ∈ 𝒜_j do
14:            if k ∈ 𝒩_p and s_temp[i] = s_temp[k] then
15:               ℒ_2 = ℒ_2 ∪ k
16:               𝒩_p = 𝒩_p\{k}
17:               s[k] = c
18:            end if
19:         end for
20:      end for
21:      ℒ_1 = ℒ_2
22:   end while
23:   c = c + 1
24: end while
```

The list $\mathcal{N}_p$ keeps track of nodes that have not yet been assigned a region number. The algorithm is finished when $\mathcal{N}_p \neq \emptyset$. To connect all adjacent nodes with the same permeability range, the lists $\mathcal{L}_1$ and $\mathcal{L}_1$ are used in a so called level-set traversal method [34]. Each iteration of the while loop in lines 10 - 22 connects the nodes in the current list to the adjacent nodes that have not been visited yet and are in the same permeability range. When $\mathcal{L}_1$ is empty at the end of the while loop, no more nodes could be merged to the current region. A new region is then initialized by picking a new node $i \in \mathcal{N}_p$.

In Algorithm 10 use Matlab code to describe a replace function. The statement $s(s = j) = i$ replaces all values j in s by i. In C++, the same can be achieved with std::replace.

Algorithm 10 Subdomain-levelset partitioner: merge

1: Input: $A \in \mathbb{R}^{n_c \times n_c}$, $s \in \mathbb{R}^{n_c \times 1}$, $p \in \mathbb{R}^{n_c \times 3}$ and $n_d \in \mathbb{R}$
2: Output: $s \in \mathbb{R}^{n_c \times 1}$
3: Choose ω and ω_+
4: $n_r = \max(s) + 1$
5: **while** $n_r > n_d$ **do**
6: $\beta =$ true
7: **while** $\beta =$ true **do**
8: $\beta =$ false
9: $\mathcal{N}_r = \{1, 2, \ldots, n_r\}$
10: Call $W =$ jumps(A, s, p, n_r)
11: **while** $\mathcal{N}_r \neq \emptyset$ **do**
12: Pick any $i \in \mathcal{N}_r$
13: $\mathcal{N}_r = \mathcal{N}_r \backslash \{i\}$
14: **for all** $j \in \mathcal{W}_i$ **do**
15: **if** $j \in \mathcal{N}_r$ **and** $W[i, j] < \omega$ **then**
16: $\mathcal{N}_r = \mathcal{N}_r \backslash \{j\}$
17: $s(s = j) = i$
18: $\beta =$ true
19: **end if**
20: **end for**
21: **end while**
22: Call $s =$renumber(s)
23: $n_r = \max(s) + 1$
24: **if** $n_r \leq n_d$ **do break end if**
25: **end while**
26: $\omega = \omega + \omega_+$
27: **end while**

After fixing the threshold ω, all region pairs with a summed permeability difference smaller than ω are merged. Because numbers are replaced in line 17, gaps will appear in the numbering. We require an ascending numbering, so renumber is used to restore the ordering in s. The implementation of the renumbering algorithm strongly depends on the programming language. Our C++ implementation is given below.

```
1 // Compute number of unique elements in s
2 st = s;
3 std::sort(st.begin(), st.end());
4 st.erase(std::unique(st.begin(), st.end()), st.end());
5 nr = st.size();
```

```
6
7  // Reorder s
8  m = 0;
9  while (m < nr)
10 {
11   smin = *std::max_element(s.begin(),s.end());
12   for(int i = 0; i < s.size(); i++)
13   {
14     if (s[i] >= m && s[i] < Pmin)
15       smin = s[i];
16   }
17   std::replace(s.begin(),s.end(), smin, m);
18   m++;
19 }
20 nr = *std::max_element(s.begin(),s.end()) + 1;
```

The renumbering process ensures that the regions are numbered in ascending order, without gaps.

Algorithm 11 is used to compute W.

Algorithm 11 Subdomain-levelset partitioner: jumps

Input: $A \in \mathbb{R}^{n_c \times n_c}$, $s \in \mathbb{R}^{n_c \times 1}$, $p \in \mathbb{R}^{n_c \times 3}$ and $n_r \in \mathbb{R}$
Output: $W \in \mathbb{R}^{n_r \times n_r}$
Initialize $W \in \mathbb{R}^{n_r \times n_r}$
for $\tilde{i} = 1, 2, \ldots,$length$(s)$ **do**
  $i = s[\tilde{i}]$
  Retrieve κ_i^x, κ_i^y and κ_i^z
  for all $\tilde{j} \in \mathcal{A}_i$ **do**
    $j = s[\tilde{j}]$
    if $i \neq j$ **then**
      Retrieve κ_j^x, κ_j^y and κ_k^z
      if $\kappa_i^x \neq \kappa_j^x$ **then**
        $W[i][j] = W[i][j] + |p_x[i] - p_x[j]|$
      else if $\kappa_i^y \neq \kappa_j^y$ **then**
        $W[i][j] = W[i][j] + |p_y[i] - p_y[j]|$
      else if $\kappa_i^z \neq \kappa_j^z$ **then**
        $W[i][j] = W[i][j] + |p_z[i] - p_z[j]|$
      end if
    end if
  end for
end for

The x, y and z coordinates of node i and j are retrieved from the IX engine, and provide a means to identify the direction of the connection between the two nodes. For example,

if the x coordinate of node i and j is different, we add $|p_x[i] - p_x[j]|$ to the (i, j)'th element of the jump matrix W.

A-1 IX partitioner result

We apply the partitioner in IX using 8 processors.

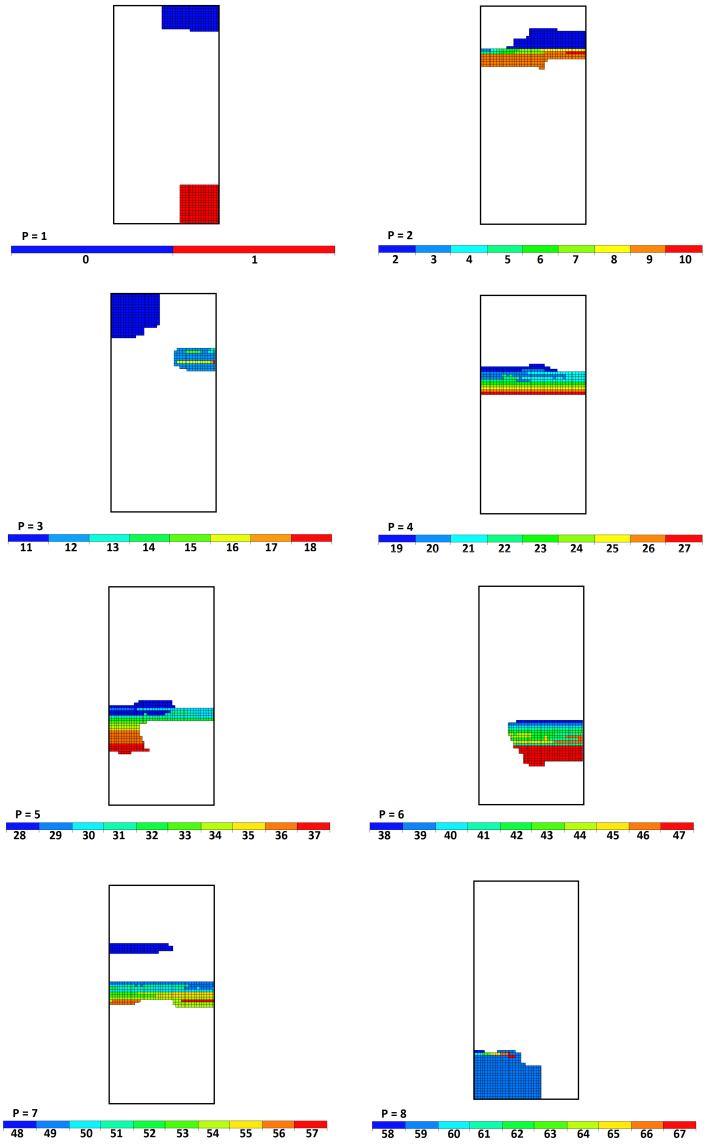


Figure A-1.1: Deflation vectors for each parallel subdomain P.

Bibliography

[1] Edwards, D. A., Gunasekera, D., Morris, J., Shaw, G., Shaw, K., Walsh, D., Fjerstad, P. A., Kikani, J., Franco, J., Hoang, V., & Quettier, L. (2011). Reservoir simulation: keeping pace with oilfield complexity. *Oilfield Review, 23*(4), 4-15.

[2] Schlumberger, Chevron & Total. (2013). Intersect Version 2013.1 technical description. Houston, TX.

[3] Cao, C., Crumpton, P. I., & Schrader, M. L. (2009, February). Efficient general formulation approach for modeling complex physics. *Paper 119165 presented at SPE Reservoir Simulation Symposium.* The Woodlands, TX: Society for Petroleum Engineers.

[4] Saad, Y. (2003). *Iterative methods for sparse linear systems* (2nd ed.). Philadelphia, PA: Society for Industrial and Applied Mathematics.

[5] G.W. Stewart. (1998). *Matrix algorithms, volume II: eigensystems.* Philadelphia, PA: Society for Industrial and Applied Mathematics.

[6] Auzinger, W., & Melenk, J. M. (2011). *Iterative solution of large linear systems.* Vienna University of Technology.

[7] Van der Vorst, H. A., & and Vuik, C. (1993). The superlinear convergence behaviour of GMRES. *Journal of Computational and Applied Mathematics, 48*, 327-341.

[8] Frank, J., & Vuik, C. (2001). On the construction of deflation-based preconditioners. *SIAM J. Sci. Comput, 23*(2), 442-462.

[9] Burrage, K. & Erhel, J. (1998). On the performance of various adaptive preconditioned GMRES strategies. *Numer. Linear Alg. Appl., 5*, 101-121.

[10] Erhel, J., Burrage, K. & Pohl, B. (1996). Restarted GMRES preconditioned by deflation. *J. of Comput. and Appl. Math., 69*, 303-318.

[11] Vuik, C., Segal, A., & Meijerink, J. (1999). An efficient preconditioned CG method for the solution of a class of layered problems with extreme contrasts of coefiňĄcients. *J. of Comp. Phys., 152*, 385âĂŞ403.

[12] Tang, J. M. (2008). Two-level preconditioned conjugate gradient methods. (Doctoral dissertation). Delft University of Technology. Retrieved from http://repository.tudelft.nl/view/ir/uuid%3Ae8c5f63b-ee7d-4a59-90daa8025f5f88b0/

[13] Jönsthövel, T. B. (2012). The deflated preconditioned conjugate gradient method, applied to composite materials. (Doctoral dissertation). Delft University of Technology. Retrieved from http://repository.tudelft.nl/view/ir/uuid%3A1c3b5b70-8e30-42d5-b55b-522b7de00abf/

[14] Klie, H., Stueben, K., Clees, T., & Wheeler, M. F. (2007, February). Deflation AMG Solvers for Highly Ill Conditioned Reservoir Simulation Problems. *Paper 105820 presented at SPE Reservoir Simulation Symposium.* Houston, TX: Society for Petroleum Engineers.

[15] Yeung, M. C., Tang, J. M., & Vuik, C. (2010). *On the convergence of GMRES with invariant-subspace deflation.* Delft University of Technology. Retrieved from http://repository.tudelft.nl/view/ir/uuid%3Af21da1b4-d4ed-4e46-a604-e1a9bdef70de/

[16] Chapman, A. & Saad, Y. (1997). Deflated and augmented Krylov subspace techniques. *Numer. Linear Alg. Appl., 4*, 43-66.

[17] Morgan, R. B. (2002). GMRES with deflated restarting. *SIAM J. Sci. Comput., 24*(1), 20-37.

[18] Wu, K., & Simon, H. (2000). Thick-restart Lanczos method for symmetric eigenvalue problems. *SIAM J. Matrix Anal. Appl., 22*, 602-616.

[19] Clemens, M., Wilke, M., Schuhmann, R., & Weiland, T. (2004). Subspace projection extrapolation scheme for transient field simulations. *IEEE Transactions on magnetics, 40*(2), 934-937.

[20] Carpenter, M. H., Vuik, C., Lucas, P., van Gijzen, M. B., & Bijl, H. (2010). A General Algorithm for Reusing Krylov Subspace Information. I. Unsteady Navier-Stokes. NASA Langley Research Center report TM2010216190. Hampton, VA: NASA.

[21] Killough, J. E. & Kossack., C.A. (1987). Fifth Comparative Solution Project: Evaluation of Miscible Flood Simulators. *Paper 16000 presented at the Ninth SPE Symposium on Reservoir Simulation.* San Antonio, TX: Society for Petroleum Engineers.

[22] Akram, F. (2010). Reservoir simulation optimizes SAGD. *The American Oil and Gas Reporter*. Retrieved from http://www.slb.com/ /media/Files/industry_challenges/heavy_oil/ industry_articles/20100_sagd_simulation.pdf

[23] Dostál, Z. (1988). Conjugate gradient method with preconditioning by projector. *Int. J. Comput. Math., 23*, 315 - 323.

[24] Nicolaides, R.A. (1987). Deflation of conjugate gradients with applications to boundary value problems. *SIAM J. Numer. Anal., 24*, 355-365.

[25] De Gersem, H., & Hameyer, K. (2000). A deflated iterative solver for magnetostatic finite element models with large differences in permeability. *Eur. Phys. J. Appl. Phys., 13*, 45-49.

[26] Tang, J. M., & Vuik, C. (2007, September). Acceleration of preconditioned Krylov solvers for bubbly flow problems. In R. Wyrzykowski and J. Dongarra and K. Karczewski and J. Wasniewski (Eds.), *Parallel Processing and Applied Mathematics 7th International Conference Proceedings* (pp. 1323-1332). Gdansk, Poland: LNCS.

[27] Nabben, R. & Vuik, C. (2008). A comparison of abstract versions of deflation, balancing and additive coarse grid correction preconditioners. *Numer. Linear Algebra Appl., 15*, 355-372.

[28] Tang, J. M., & Vuik, C. (2007). New variants of deflation techniques for pressure correction in bubbly flow problems. *Journal of Numerical Analysis, Industrial and Applied Mathematics, 2*, 227-249.

[29] Tang, J. M., & Vuik, C. (2007). Efficient deflation methods applied to 3-D bubbly flow problems. *Electronic Transactions on Numerical Analysis, 26*, 330-349.

[30] Tang, J. M., & Vuik, C. (2007). On deflation and singular symmetric positive semi-definite matrices. *Journal of Computational and Applied Mathematics, 206*, 603-614.

[31] MacLachlan, S. P., Tang, J. M., & Vuik, C. (2008). Fast and robust solvers for pressure correction in bubbly flow problems. *Journal of Computational Physics, 227*, 9742-9761.

[32] Tang, J. M., MacLachlan, S. P., Nabben, R., & Vuik, C. (2010). A comparison of two-Level preconditioners based on multigrid and deflation. *SIAM. J. Matrix Anal. and Appl., 31*, 1715-1739.

[33] Jönsthövel, T. B., van Gijzen, M. B., Vuik, C. & Scarpas, A. (2013) On the use of rigid body modes in the deflated preconditioned conjugate gradient method. *SIAM J. Sci. Comput., 35*, B207-B225.

[34] Lingen, F. J., Bonnier, P. G., Brinkgreve, R. B. J., van Gijzen, M. B. & Vuik, C. (2012) A parallel linear solver exploiting the physical properties of the underlying mechanical problem (Delft Institute of Applied Mathematics report 12-12). Delft University of Technology.

[35] Jönsthövel, T. B., Gijzen, M. B., MacLachlan, S., Vuik, C. & Scarpas, A. (2012). Comparison of the deflated preconditioned conjugate gradient method and algebraic multigrid for composite materials. *Computational Mechanics, 50*, 321-333.

[36] Jönsthövel, T. B., van Gijzen, M. B., Vuik, C., Kasbergen, C., & Scarpas, A. (2009). Preconditioned conjugate gradient method enhanced by deflation of rigid body modes applied to composite materials. *Computer Modeling in Engineering and Sciences, 47*, 97-118.

[37] van 't Wout, E., van Gijzen, M. B., Ditzel, A., van der Ploeg, A. & Vuik, C. (2010) The deflated relaxed incomplete Cholesky CG method for use in a real-time ship simulator. *Procedia Computer Science, 1*, 249-257.

[38] Vuik, C., & Frank, J. (2000, August). Deflated ICCG method applied to problems with extreme contrasts in the coefficients. In Deville, M. & Owens, R. (Eds.), *Proceedings of the 16th IMACS World Congress 2000*, CDROM ISBN 3-9522075-1-9. New Brunswick: Rutgers University.

[39] Vuik, C., Segal, A., Meijerink, J.A., & Frank, J. (2000, April). Deflated ICCG applied to problems with extreme contrasts in the coefficients. In Manteuffel, T. A., & McCormick, S. F. (Eds.), *Sixth Copper Mountain Conference on Iterative Methods*, Copper Mountain, CO.

[40] Vuik, C., Segal, A., Meijerink, J.A., & Wijma, G.T. (2000). The construction of projection vectors for a Deflated ICCG method applied to problems with extreme contrasts in the coefficients. Shell Report EP2000-8019. Den Haag: Shell.

[41] Vuik, C., Segal, G., & Meijerink, K. (1998, March). An efficient CG method for layered problems with large contrasts in the coefficients. In Manteuffel, T. A., & McCormick, S. F. (Eds.), *Fifth Copper Mountain Conference on Iterative Methods*, Copper Mountain, Colorado.

[42] Lee, S. H., Zhou, H., & Tchelepi, H. A. (2009). Adaptive fully implicit multi-scale finite-volume method for multi-phase flow and transport in heterogeneous formations. *Journal of Computational Physics, 228*(24), 9036 - 9058.

[43] Cao, H., Tchelepi, H. A., Wallis, J., & Yardumian, H. (2005, October). Parallel scalable unstructured CPR-type linear solver for reservoir simulation. *Paper 96809 presented at the SPE Anuual Technical Conference and Exhibition.* Dallas, TX: Society for Petroleum Engineers.

[44] von Mises, R. & Pollaczek-Geiringer, H. (1929). Praktische Verfahren der Gleichungsauflösung. *Zeitschrift für Angewandte Mathematik und Mechanik, 9*, 152-164.

[45] Greenbaum, A. Pták, V. & Strakos, Z. (1996). Any Nonincreasing Convergence Curve is Possible for GMRES. *SIAM J. Matrix Anal. Appl., 17*, 465-469.

[46] Saad, Y., & Schultz, M. H. (1986). GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput., 7*, 856-869.

[47] Liesen, J., & Tichý, P. (2004). Convergence analysis of Krylov subspace methods. *GAMM Mitt. Ges. Angew. Math. Mech., 27*(2), 153 - 173.

[48] Van der Vorst, H.A., and Vuik, C. (1993). The superlinear convergence behaviour of GMRES. *Journal of Computational and Applied Mathematics, 48*, 327-341.

[49] Greenbaum, A., & Trefethen, L. N. (1994). GMRES/CR and Arnoldi/Lanczos as matrix approximation problems. *SIAM J. Sci. Comput., 15*, 359âĂŞ368.

[50] Trefethen, L. N. (1990). Approximation theory and numerical linear algebra, in Algorithms for approximation, II (Shrivenham, 1988), 336-360. London: Chapman and Hall.

[51] Nachtigal, N. M., Reddy, S. C., & Trefethen, L. N. (1992). How fast are nonsymmetric matrix iterations? *SIAM J. Matrix Anal. Appl., 13*, 778âĂŞ795.

[52] Eiermann, M., & Ernst, O. G. (2001). Geometric aspects of the theory of Krylov subspace methods. *Acta Numer., 10*, 251-312.

[53] Nevanlinna, O. (1993). Convergence of iterations for linear equations. *Lectures in Mathematics ETH Zurich, Birkhäuser Verlag.* Basel: ETH Zurich.

[54] Morgan, R. B. (1995). A restarted GMRES method augmented with eigenvectors. *SIAM J. Matrix Anal. Appl., 16*, 1154-1171.

[55] Morgan, R. B. (2000). Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations. *SIAM J. Matrix Anal. Appl., 21*, 1112-1135.

[56] Kharchenko, S. A. & Yeramin, A. (1995). Eigenvalue translation based preconditioners for the GMRES(k) method. *SIAM J. Matrix Anal. Appl., 2*, 51-77.

[57] Baglami, J., Calvetti, D., Golub, G.H., & Reichel, L. Adaptively preconditioned GMRES algorithms. Technical report, Kent State University.

[58] Nabben, R. & Vuik, C. (2004). A comparison of deflation and coarse grid correction applied to porous media flow. *SIAM J. Numer. Anal., 42*, 1631-1647.

[59] Mansfield, L. (1990). On the conjugate gradient solution of the Schur complement system obtained from domain decomposition. *SIAM J. Numer. Anal., 27*, 1612-1620.

[60] Mansfield, L. (1991). Damped Jacobi preconditioning and coarse grid deïňĆation for conjugate gradient iteration on parallel computers. *SIAM J. Sci. Statist. Comput., 12*, 1314-1323.

[61] Vuik, C., Segal, A., L. el Yaakoubim E., & Dufour, B. (2002). A comparison of various deflation vectors applied to elliptic problems with discontinuous coefficients. *Applied Numerical Mathematics, 41*, 219-233.

[62] Vermolen, F., Vuik, C., & Segal, A. (2004). Deflation in preconditioned Conjugate Gradient methods for finite element problems. In Krizek, M., Neittaanmaki, P., Glowinski, R., & Korotov, S. (Eds.). (2004). Conjugate gradient and finite element methods (pp. 103-129). Berlin: Springer.

[63] Padiy, A., Axelsson, O. Polman, B. (2000). Generealized augmented matrix preconditioning approach and its application to iterative solution of ill-conditioned algebraic systems. *SIAM J. Matrix Anal. Appl, 22*, 793-818.

[64] Wallis, J. R. (1983, November). Incomplete gaussian elimination as a preconditioning for generalized conjugate gradient acceleration. *Paper 12265 presented at SPE Reservoir Simulation Symposium.* San Fransisco, CA: Society for Petroleum Engineers.

[65] Wallis, J. R., Kendall, R. P., Little, T. E., & Nolen, J. S. (1985, February). Constrained residual acceleration of conjugate residual methods. *Paper 13536 presented at SPE Reservoir Simulation Symposium.* Dallas, TX: Society for Petroleum Engineers.

[66] Cao, H., Tchelepi, H. A., Wallis, J. R., & Yardumian, H. (2005, October). Parallel scalable CPR-type linear solver for reservoir simulation. *Paper 96809 presented at SPE Annual Technical Conference and Exhibition.* Dallas, TX: Society for Petroleum Engineers.

[67] Cao, H. (2002, June). Development of techniques for general purpose reservoir simulators. (Doctoral dissertation). Stanford University. Retrieved from http://edces.netne.net/files/Cao_Thesis_40469a.pdf

[68] Trottenberg, U., Oosterlee, C., & Schüller, A., Guest contributions by Stüben, K., Oswald, Pl, Brandt, A. (2001). Multigrid. San Diego, CA: Academic Press.

[69] Ruge, J. W., & Stüben, K. (1987). Algebraic multigrid (AMG). In S.F. McCormick (Ed.), Multigrid Methods, volume 3 of Frontiers in Applied Mathematics (73-130). Philadelphia, PA: Society for Industrial and Applied Mathematics.

[70] Brandt, A., McCormick, S. F., & Ruge, J. W. (1982). Algebraic multigrid (AMG) for sparse matrix equations. In D.J. Evans (Ed.), Sparsity and Its Applications (257-284). Cambridge: Cambridge University Press.

[71] Dendy, J. E. (1982). Black box multigrid. *J. Comput. Phys., 48*(3), 366-386.

[72] Clees, T. (2005). AMG strategies for PDE systems with applications in industrial semiconductor simulation. (Doctoral dissertation). Universität zu Köln, Köln. Retrieved from http://www.scai.fraunhofer.de/fileadmin/download/samg/paper/Clees_Diss.pdf

[73] Stüben, K., Clees, T., Schneider, M., Klie, H., Lou, B. & Wheeler, M. (2007, February). Algebraic Multigrid Methods (AMG) for the Efficient Solution of Fully Implicit Formulations in Reservoir Simulation. *Paper 105832 presented at the SPE Reservoir Simulation Symposium.* Houston, TX: Society for Petroleum Engineers.

[74] Füllenbach, T., & Stüben, K.. (2002). Algebraic Multigrid for Selected PDE Systems. In Rolduc and Gaeta (Eds.), Proceedings of the 4th European Conference (399-410). London, NJ: World ScientiiňĄc.

[75] Jiang, Y. (2007). Techniques for modeling complex reservoirs and advanced wells. (Doctoral dissertation). Stanford University. Retrieved from https://pangea.stanford.edu/ERE/pdf/pereports/PhD/Jiang07.pdf.

[76] Brandt, A. (1986). Algebraic multigrid theory: the symmetric case. *Appl. Math. Comp., 19*, 23-56.

[77] van der Vorst, H. (2003). Iterative Krylov methods for large linear systems. Cambridge University Press.

[78] Georqe, A., & Liu, J. W. H. (1981). Computer Solution of Large Sparse Positive Definite Systems. Englewood Cliffs, N.J.: Prentice Hall, Inc.

[79] Tang, J.M. (2005). Parallel deflated CG methods applied to linear systems from moving boundary problems. Technical report, Delft University of Technology. Retrieved from http://repository.tudelft.nl/view/ir/uuid%3Adbd4809c-784b-4da4-82bb-3f7a980ce90c/

[80] Aksoylu, B., Klie, H., & Wheeler, M.F. (2007). Physics-based preconditioners for porous media flow applications. ICES Technical Report. The University of Texas at Austin.

[81] Karypis, G. & Kumar, V. (1997). A coarse-grain parallel formulation of multilevel k-way graph-partitioning algorithm. *In Proc. 8th SIAM Conference on Parallel Processing for Scientific Computing.*

[82] Klie, H., Monteagudo, J., Hoteit, H., & Rodriguez, A. (2009, February). Towards a new generation of physics-driven solvers for black-oil and compositional flow simulation. *Paper 118752 presented at SPE Reservoir Simulation Symposium.* The Woodland, TX: Society for Petroleum Engineers.