



university of
 groningen

faculty of mathematics
 and natural sciences

Deflation-based preconditioning in the THCM model

Leo van Kampenhout

Bachelor Thesis in Applied Mathematics

Supervisor(s): J. Thies, F.W. Wubs

August 2008

Deflation-based preconditioning in the THCM model

Leo van Kampenhout

Supervisor(s):

J. Thies, F.W. Wubs

Institute of Mathematics and Computing Science

P.O. Box 407

9700 AK Groningen

The Netherlands

Contents

1	Introduction	1
1.1	Model description	1
1.2	Problem description	2
1.2.1	The Jacobian matrix	2
1.2.2	A new preconditioner using deflation?	3
1.2.3	The dataset	3
1.3	Outline of this thesis	4
2	About deflation	5
2.1	Introduction	5
2.2	Deflation vectors	5
2.3	Deflation for the non-symmetric case	6
2.4	Implementation	7
3	Deflation on the entire system	9
3.1	Eigenvalues	9
3.2	Setup	9
3.3	Results	10
4	Deflation on the Schur problem	11
4.1	Structure	11
4.2	Setup	12
4.3	Results	13
4.4	About runtimes	13
5	Deflation on the simplified Schur problem	15
5.1	Eigenvalues	15
5.2	Setup and results	15
6	Conclusions	19
6.1	Deflation	19
6.2	Recommendations for further study	20

Chapter 1

Introduction

1.1 Model description

The computermodel THCM has been developed at the Institute of Marine and Atmospheric research Utrecht (IMAU), in collaboration with the University of Groningen. The aim is to model three dimensional *thermohaline* ocean flows. These flows are primarily driven by heat- and freshwater fluxes and regulate heat transfer around the globe. Therefore, a possible application of this model is climate change prediction.

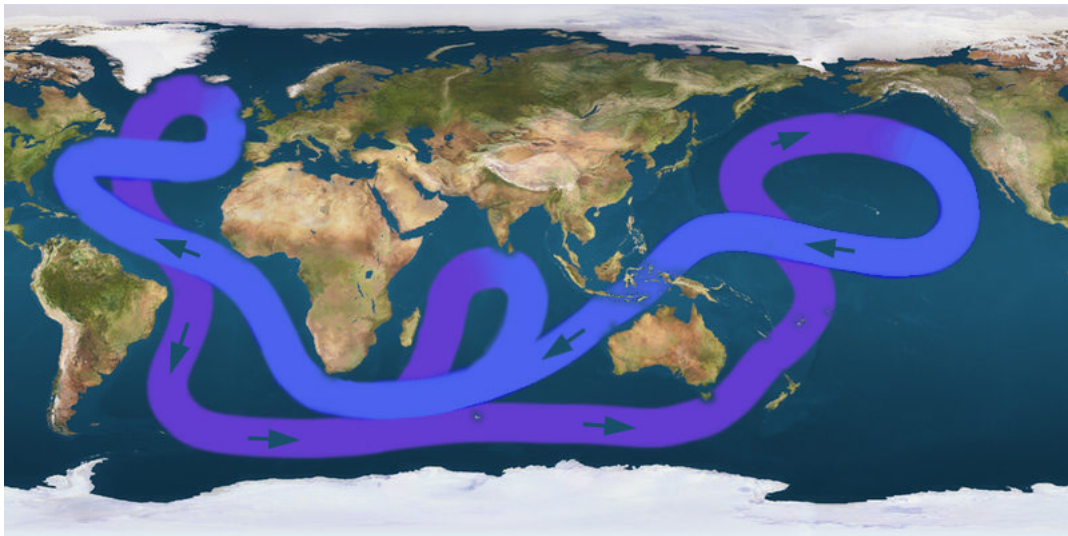


Figure 1.1: Thermohaline ocean circulation (from [6]).

An implicit discretization scheme is used to allow for large time steps. Time steps of 10-100 years are possible. The drawback of implicit schemes is that each time step requires the solution of a system of nonlinear equations. In order to solve these systems, continuations methods [1] or Jacobian Free Newton Krylov [7] methods are used.

In June 2007 the PhD dissertation *Solving Large Linear Systems in an Implicit Thermohaline Ocean Model* by Arie de Niet was published. This thesis proposes a new procedure for solving large linear systems. In a typical continuation process, several hundred such linear systems have to be solved. It turned out that the solution of these systems is the bottle neck

in the computations.

The most important feature introduced by De Niet is exploitation of the structure of the system. He reveals dependency of certain variables by transforming the equations, resulting in a near upper block-triangular system. He then proposes a solver for these transformed equations.

In our research we investigate a method which might increase the speed of this linear system solver.

1.2 Problem description

In this section we look closer to the linear systems we have to solve. We solve these systems on a certain domain-grid. How this domain was chosen is also explained in this chapter.

1.2.1 The Jacobian matrix

Computing thermohaline ocean flows involves a number of variables. In the THCM model these are: the flow velocity (in three directions), pressure, temperature and salinity. The physical relations between these quantities are described by a system of nonlinear PDEs.

These nonlinear equations are solved using numerical continuation. In this process, large linear systems have to be solved. To be precise, linear systems arise in the tangent predictor step for a subsequent Newton solve. The coefficient matrix of the linear system is therefore a Jacobian matrix. Typically the linear system looks like

$$\begin{bmatrix} \mathbf{A}_{uv} & \mathbf{E}_{uv} & \mathbf{G}_{uv} & 0 \\ 0 & 0 & \mathbf{G}_w & \mathbf{B}_{TS} \\ \mathbf{D}_{uv} & \mathbf{D}_w & 0 & 0 \\ \mathbf{B}_{uv} & \mathbf{B}_w & 0 & \mathbf{A}_{TS} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{uv} \\ \mathbf{u}_w \\ \mathbf{u}_p \\ \mathbf{u}_{TS} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{uv} \\ \mathbf{b}_w \\ \mathbf{b}_p \\ \mathbf{b}_{TS} \end{bmatrix}. \quad (1.1)$$

Each \mathbf{G}_* represents a discrete gradient operator and each \mathbf{D}_* a discrete divergence operator. The matrices \mathbf{A}_{TS} and \mathbf{A}_{uv} are of convection-diffusion type. The matrices \mathbf{B}_* represent the couplings between the dynamic and thermodynamic variables. The matrix \mathbf{E}_{uv} represents the coupling to w in the material derivative. However this block will be ignored and replaced by zeroes, for it's entries are very small compared to the others. De Niet rewrites the system as (cf. eq.7.10 [1])

$$\begin{bmatrix} \mathbf{A}_p & 0 & 0 & \mathbf{B}_{TS} \\ \hat{\mathbf{G}}_{uv} & \mathbf{K}_{uv\bar{p}} & 0 & 0 \\ 0 & \hat{\mathbf{D}}_{uv} & \mathbf{A}_w & 0 \\ 0 & \hat{\mathbf{B}}_{uv} & \mathbf{B}_w & \mathbf{A}_{TS} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\bar{p}} \\ \mathbf{u}_{uv\bar{p}} \\ \mathbf{u}_w \\ \mathbf{u}_{TS} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_w \\ \mathbf{b}_{uv\bar{p}} \\ \mathbf{b}_{\bar{p}} \\ \mathbf{b}_{TS} \end{bmatrix}. \quad (1.2)$$

The coefficient matrix we see is the transformed and permuted Jacobian matrix, which we will denote by Φ .¹ We can now construct the following block LU factorization for Φ :

¹De Niet: $\hat{\Phi}$

$$\mathbf{L}_\Phi \mathbf{U}_\Phi = \begin{bmatrix} \mathbf{A}_p & 0 & 0 & 0 \\ \hat{\mathbf{G}}_{uv} & \mathbf{K}_{uv\bar{p}} & 0 & 0 \\ 0 & \hat{\mathbf{D}}_{uv} & \mathbf{A}_w & 0 \\ 0 & \hat{\mathbf{B}}_{uv} & \mathbf{B}_w & \mathbf{S}_{TS} \end{bmatrix} \begin{bmatrix} \mathbf{I}_p & 0 & 0 & \mathbf{A}_p^{-1} \mathbf{B}_{TS} \\ 0 & \mathbf{I}_{uv+\bar{p}} & 0 & -\mathbf{K}_{uv\bar{p}}^{-1} \hat{\mathbf{G}}_{uv} \mathbf{A}_p^{-1} \mathbf{B}_{TS} \\ 0 & 0 & \mathbf{I}_w & \mathbf{A}_w^{-1} \hat{\mathbf{D}}_{uv} \mathbf{K}_{uv\bar{p}}^{-1} \hat{\mathbf{G}}_{uv} \mathbf{A}_p^{-1} \mathbf{B}_{TS} \\ 0 & 0 & 0 & \mathbf{I}_{TS} \end{bmatrix},$$

where the Schur complement is given by

$$\mathbf{S}_{TS} = \mathbf{A}_{TS} + (\hat{\mathbf{B}}_{uv} - \hat{\mathbf{B}}_w \mathbf{A}_w^{-1} \hat{\mathbf{D}}_{uv}) \mathbf{K}_{uv\bar{p}}^{-1} \hat{\mathbf{G}}_{uv} \mathbf{A}_p^{-1} \mathbf{B}_{TS}. \quad (1.3)$$

This factorization is exact. But as De Niet points out, in practice it is impossible to compute this Schur complement, for the matrix will be dense. There are two possibilities:

1. Forget about the Schur complement and use $\mathbf{S}_{TS} = \mathbf{A}_{TS}$. In this case it is of no use to apply the block \mathbf{U}_Φ . This is the same as ignoring the block \mathbf{B}_{TS} in (1.2), which leads to a block Gauss-Seidel preconditioner, which we will refer to as the BGS preconditioner.
2. We solve the equation $\mathbf{S}_{TS} \cdot \mathbf{y}_{TS} = \mathbf{b}_{TS}$ via an iterative procedure using a Krylov subspace method. In this case we only require the matrix \mathbf{S}_{TS} to be applied to a vector, which is relatively cheap. This results in a kind of block incomplete LU factorization, which we will call the BILU preconditioner.

1.2.2 A new preconditioner using deflation?

As De Niet points out in his PhD dissertation, the BILU preconditioner is too slow to be used in practice. Even with some approximations made (he does not tell us which approximations), BGS is by far superior over BILU [1, page 109]. For now we will take this as a fact.

Deflation is a technique to speed up numerical solvers, and can be used in combination with preconditioners. The question that initiated our research is the following. Is it possible to construct a preconditioner based on BILU and deflation, that is faster than BGS?

However, in the research we consider deflation more generally than this. If the proposed preconditioner would not meet the expectations – can we perhaps apply deflation to BGS instead? Or should we not look to the Schur-problem alone – can deflation be applied to the entire Jacobian system?

1.2.3 The dataset

Since the main purpose is to test the *viability* of the deflation method, we do not bother with the original FORTRAN code used in THCM. Instead we use a simplified MATLAB-code, which works on a smaller dataset. In the first part of the thesis, we make use of the dataset `small_weak`. Later we will also consider more realistic datasets, dataset `small_strong` and dataset `big_strong`. These all are discretizations of the physical equations, cf. De Niet [1, chapter 2]. The domain is the Atlantic Ocean, to be precise:

- the domain is a single-hemispheric basin 10-74 degrees north, 286-350 degrees longitude, 4000 meters deep.
- the domain has idealized geometry, i.e. no continents.

- the grid-size is 16x16x16 grid, corresponding to 4 degrees resolution. In the dataset `big_strong`, grid-size is 32x32x16 and has 2 degrees resolution.
- the flow is forced by a wind field from measurements and sine-shaped salt- and temperature fields, all of them premultiplied by a continuation parameter γ . The ‘weak’ test-case is at $\gamma = 0.1$, the ‘strong’ ones at $\gamma = 1.0$.

The test problem is one linear system with the Jacobian of the nonlinear system. Bear in mind that in the numerical continuation process, several hundreds of such systems have to be solved.

Since there are 6 variables in the model – speed in 3 directions, temperature, pressure and salinity – we have to solve 24576 unknowns in the `small_weak` and `small_strong` datasets. In the dataset `big_strong` we have to solve 98304 unknowns.

1.3 Outline of this thesis

In the next chapter we will treat the deflation technique and our implementation of it. In the third chapter we will apply deflation on the entire Jacobian system. In the fourth chapter, we will apply deflation on the Schur system \mathbf{S}_{TS} . The preconditioner in that chapter will be called BILU-D. In the fifth chapter we will apply deflation on the simplified Schur system \mathbf{A}_{TS} system, giving rise to the BGS-D preconditioner. We finish with conclusions and recommendations for further study.

Chapter 2

About deflation

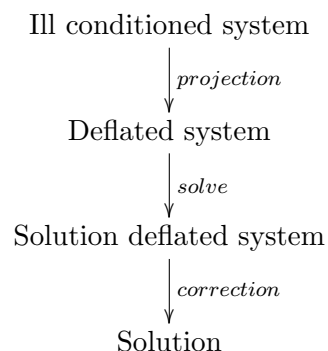
In this chapter we will first give a short introduction to deflation. Next we will look at deflation for the non-symmetric case and give insight about how we have implemented this in our code.

2.1 Introduction

The deflation method was originally proposed by Nicolaides [5] in 1987, as a method for improving the convergence of the conjugate gradient (CG) method. It could be used stand-alone, or in conjunction with preconditioning.

The method has proven successful when there are a few isolated extremal eigenvalues. It constructs a so-called *projection* matrix, which projects the extremal eigenvalues out of the problem. By doing so, the effective condition number of the problem is reduced, which usually makes the solver use fewer iterates than before. For bounds on the condition number, consult the article on deflation-based preconditioners written by J. Frank and C. Vuik [2]. We will use the results of their article as a motivation for the rest of this thesis.

The idea behind deflation can be pictured as follows.



2.2 Deflation vectors

The difficulty of the deflation method lies in the choice of the *deflation vectors*, which generate the projection matrix. The deflation vectors pose the real challenge. A good choice of deflation

vectors leads to good convergence, a bad choice will not help us at all. In general there are two ways to choose deflation vectors.

The first one is called *subdomain deflation*, since it uses subdomains to construct the deflation vectors. The subdomains can be chosen arbitrary, but as Frank and Vuik point out, there are guidelines for optimal configurations. Each subdomain will usually have one corresponding deflation vector, that has the value ‘1’ on the subdomain, and ‘0’ outside.¹ This approach is related to domain decomposition and multigrid methods.

The second possibility is to take (approximate) eigenvectors as deflation vectors. We will refer to this as *eigenvector deflation*. To make the convergence speed up, the eigenvectors should represent extremal eigenvalues. The approximate eigenvectors can even be extracted from the Krylov subspace generated by GMRES – but we have not further investigated this.

In the second approach we will usually not need many deflation vectors, since we only want to remove the most extremal eigenvalues. Furthermore, when we have information about the placement of the eigenvalues, we can estimate how many deflation vectors will suffice. This in contrast to subdomain deflation, where we always need a significant amount of deflation vectors. Namely, there should be at least enough vectors to span a space in which we can approximate the extremal modes. In this way, the eigenvectors corresponding to the extremal eigenvalues will largely be projected out of the problem, as we would like to see.

Obviously we will be in trouble with subdomain deflation when the extremal modes are high frequent, making it impossible to approximate them with (a limited number of) subdomain vectors.

2.3 Deflation for the non-symmetric case

This subsection is based on Frank and Vuik, §5.1. Consider the linear system $Au = f$, where A is non-symmetric. Let the projections P and Q be given by

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

The matrices Z and Y should contain deflation vectors. As motivated by Frank and Vuik, we will take $Y = Z$. We have the following properties for P and Q :

- $P^2 = P, Q^2 = Q$.
- $PAZ = Y^T P = 0, Y^T A Q = QZ = 0$.
- $PA = AQ$

To solve the system $Au = f$ using deflation, we split the solution vector u as follows:

$$u = (I - Q)u + Qu.$$

The *projected* system $(I - Q)u = Z(Y^T AZ)^{-1}Y^T Au = Z(Y^T AZ)^{-1}f$ can be computed immediately. Furthermore, Qu can be obtained by solving the *deflated* system

$$PA\tilde{u} = Pf \tag{2.1}$$

for \tilde{u} and premultiplying the result with Q .

¹When dealing with multiple variables in one system, we might need more than one deflation vector per subdomain. For example, with THCM this is the case.

2.4 Implementation

In this thesis, we will consider two implementations of the deflation method. The first implementation is to make use of the `DGMRES` function, an adapted `GMRES` function by Bart Dopheide, 2004 [3]. The syntax of this function is:

```
function [x,flag,relres,iter,resvec] = ...
    dgmres(A,b,restart,tol,maxit,M1,M2,x0,Z,Y,varargin)
```

We will refer to this implementation as the *dgmres* solver. The advantage of this solver is that it is easy to use, since it is already implemented.

The second implementation is custom-made. Beforehand we compute and store the recurring matrix products, such as $A_c = (Y^T AZ)$ and AZ . Then we let the standard `GMRES` routine solve the deflated system (2.1). We will refer to this implementation as the *custom* solver. The advantage of the custom solver is that, in theory, it can be used in combination with any Krylov solver.

Due to different stopping criteria for both methods, they yield different results. To make it precise,

- standard `GMRES` will finish when $\|b - Au\| < \text{tol}$.
- the *dgmres* solver will finish when $\|Pb - PAu\| < \text{tol}$.
- the custom solver will finish when $\|Pb - PA\tilde{u}\| < \text{tol}$. Subsequently it will premultiply \tilde{u} with Q and add the correction term $(I - Q)u$.

As a result, the custom solver will yield a better solution using the same tolerance value. The (relative) residue will therefore be significantly smaller than the tolerance value. But to reach that solution, it will usually need more iterations than the *dgmres* solver.

Because of this, we actually should not compare the number of iterations of both solvers, since they represent different results. But we included both for completeness. Keep in mind that by changing the tolerance, for example from 10^{-3} to 10^{-2} , the custom solver might still return results with residue smaller than 10^{-3} . This is important for future work on the subject.

Chapter 3

Deflation on the entire system

In this chapter we will apply deflation to the entire Jacobian system (1.2). Although this can give problems in larger datasets, for now it can be a good indicator of how good deflation works.

3.1 Eigenvalues

We consider both subdomain and eigenvector deflation. For the latter we need to approximate eigenvectors of Φ , which is an costly business (not feasible with more realistic datasets). We computed the 16 smallest eigenvalues and their corresponding eigenvectors from the dataset `small_weak`. A plot of the eigenvalues can be seen in figure 3.1. We see that the smallest two eigenvalues are complex conjugated. It turns out that removing one eigenvalue gives exactly the same results as we would remove the pair. This can be explained by the similarity between the corresponding eigenvectors. As a matter of fact, the ‘conjugated’ eigenvectors are identical in the real part, and each others ‘negative’ in the imaginary part.

3.2 Setup

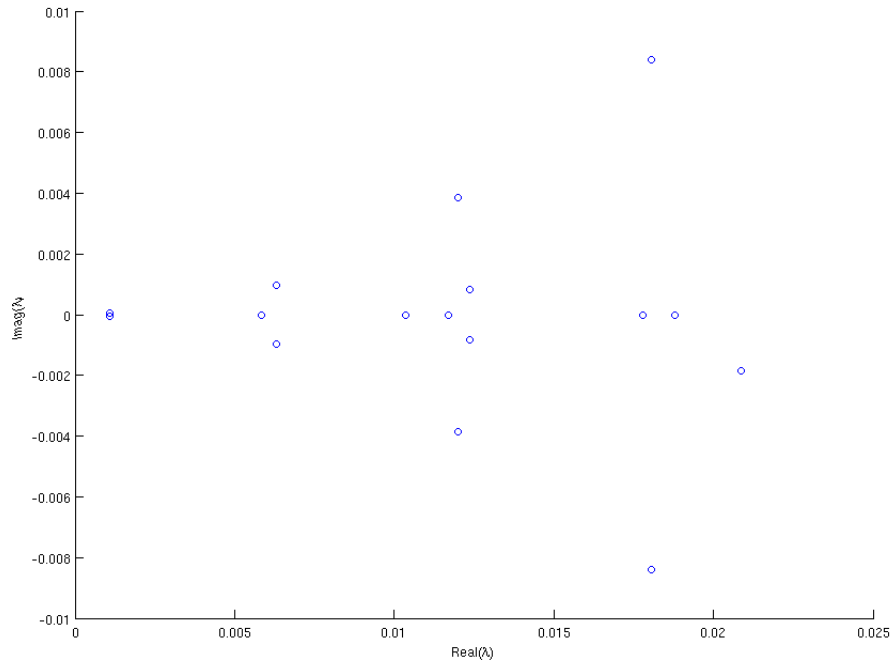
We test both deflation methods on the dataset `small_weak`, which was introduced in section 1.2.3. We use the `DGMRES` solver, introduced in section 2.4, with tolerance 10^{-9} . As preconditioner for this solver, we use an ILU decomposition of Φ (`droptol` = 10^{-3}). We don’t let the solver restart.¹

For subdomain deflation, we divide the three dimensional (cubic) grid into equal sized smaller cubes. In dimension x, y, z we divide the grid into respectively n, m, p pieces. The total number of subdomains is then nmp .

Our choice of subdomain deflation vectors is motivated by Frank and Vuik. For each subdomain we create 2 deflation vectors, let’s say z_T and z_S .² They have respectively the temperature (T) component or the salinity (S) component of the vector equal to ‘1’ and the rest of the components ‘0’. Together, these $2nmp$ vectors form/span the deflation space Z .

¹Although deflation can improve convergence after a restart. For more information, see Frank and Vuik [2], reference 19.

²One would expect 6 deflation vectors, one for each variable. However, the eigenvectors corresponding to the smallest eigenvalues have neglectable values in the other 4 components.

Figure 3.1: Plot of the 16 smallest eigenvalues of Φ .

3.3 Results

The two tables below display the total of iterations when using eigenvector resp. subdomain deflation. The column containing '-' represents the solver without deflation.

We notice that by computing only 1 eigenvector, we get rid of 4 outer iterations. That is about 15% of the total, which is not bad at all. However, computing one eigenvector for such a large system is a big expense. Note that removing 2 eigenvectors gives the same result, since those eigenvalues are a complex pair.

With subdomain deflation rather the same story. Now 32 subdomains (= 64 deflation vectors) will get rid of 5 outer iterations. The advantage here is that these deflation vectors are very easy to construct.

Deflation vectors	-	1	2	3	4	5	10	11
Outer iterations	28	24	24	23	22	22	21	21

Table 3.1: Outer iterations. Using eigenvector deflation.

Subdomains	-	8	8	16	32	32	64
Configuration	-	(2, 2, 2)	(1, 1, 8)	(2, 2, 4)	(4, 4, 2)	(2, 2, 8)	(4, 4, 4)
Outer iterations	28	26	24	25	26	23	23

Table 3.2: Outer iterations. Using subdomain deflation.

Chapter 4

Deflation on the Schur problem

We will now consider deflation on the Schur problem, i.e. deflation on the system

$$\mathbf{S}_{TS} \cdot \mathbf{u}_{TS} = \mathbf{b}_{TS},$$

with \mathbf{S}_{TS} being the Schur complement. Solving the system traditionally gives rise to the preconditioner we named BILU. Now with deflation, we hope increase convergence a bit.

4.1 Structure

First we will turn to the overall structure of the program. As solver for the entire system (1.2), we will turn to the flexible GMRES routine proposed by Saad [4]. Our implementation has been written by De Niet. It is essential to use this FGMRES routine, for we use a *dynamic* preconditioner – on them the default GMRES routine will fail.

$$\underbrace{\begin{bmatrix} \mathbf{A}_p & 0 & 0 & 0 \\ \hat{\mathbf{G}}_{uv} & \mathbf{K}_{uv\bar{p}} & 0 & 0 \\ 0 & \hat{\mathbf{D}}_{uv} & \mathbf{A}_w & 0 \\ 0 & \hat{\mathbf{B}}_{uv} & \mathbf{B}_w & \mathbf{S}_{TS} \end{bmatrix}}_{\mathbf{L}_\Phi} \underbrace{\begin{bmatrix} \mathbf{I}_p & 0 & 0 & \mathbf{U}_1 \\ 0 & \mathbf{I}_{uv+\bar{p}} & 0 & \mathbf{U}_2 \\ 0 & 0 & \mathbf{I}_w & \mathbf{U}_3 \\ 0 & 0 & 0 & \mathbf{I}_{TS} \end{bmatrix}}_{\mathbf{U}_\Phi} \begin{bmatrix} \mathbf{u}_{\bar{p}} \\ \mathbf{u}_{uv\bar{p}} \\ \mathbf{u}_w \\ \mathbf{u}_{TS} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_w \\ \mathbf{b}_{uv\bar{p}} \\ \mathbf{b}_{\bar{p}} \\ \mathbf{b}_{TS} \end{bmatrix} \quad (4.1)$$

This solver will rely on the BILU-D routine to perform the preconditioning. In every FGMRES iteration our routine will take the input-vector \mathbf{b} to construct an *approximation* of $\mathbf{L}_\Phi^{-1} \mathbf{U}_\Phi^{-1} \mathbf{b}$, the output. The better this approximation is, the fewer iterations FGMRES needs to complete. But each time we improve the approximation, we will have to pay a price.

In the preconditioner, we need to implement two steps. First step, solve the system with \mathbf{L}_Φ . Second step, solve the system with \mathbf{U}_Φ . In the first step, the fourth block equation gets special attention, since we apply deflation there.

Solving the system $\mathbf{L}_\Phi \mathbf{y} = \mathbf{b}$

Solving the first 3 row-equations involves the inverse of the matrix $\mathbf{K}_{uv\bar{p}}$. This matrix is of saddle point type, a special type of matrix occurring often in numerical fluid dynamics. The matrix can be solved using an expensive (I)LU-decomposition for $\mathbf{K}_{uv\bar{p}}$,

or a call to the `SIMPLER` routine. Since this is not relevant to our topic, we won't discuss it any further and refer to [1, chapter 6].

We also need inverses of \mathbf{A}_p and \mathbf{A}_w , but they are relatively cheap to compute.

Solving the last row-equation involves \mathbf{S}_{TS} . This is the so-called Schur problem, the system on which we intend to use deflation. This equation cannot be solved exact, hence we will use a Krylov-method. We have the possibility to decrease the tolerance of this method, obtaining better approximations, thus less outer `FGMRES` iterations. But the number of inner iterations will inevitably increase, so caution is required.

Solving the Schur problem

Without deflation, we could just call `GMRES` to solve the problem right away. `GMRES` in MATLAB requires the matrix or its *function-handle* to be given. Since we only know that \mathbf{S}_{TS} looks like (1.3), we give its function-handle. The only thing this function can do, is take a vector and apply \mathbf{S}_{TS} to it. This function can be found in the appendix (`apply_STS.m`).

With deflation, we need to do more. First of all, the deflation vectors. We observe that only subdomain deflation is feasible now, since we do not know what \mathbf{S}_{TS} looks like. Computing eigenvectors is then way too expensive, if not impossible.

We will store the subdomain vectors in the variable Z , after computing them in a subroutine, which can be found in the appendix (`create_Z.m`). Next we can compute $AZ = \mathbf{S}_{TS} \cdot Z$ and $A_c = Z^T \cdot AZ$, which we will need in our projections P and Q . Since Z remains unchanged throughout the whole runtime, it is possible to compute these things beforehand, saving valuable CPU-cycles. Recall that when using deflation the solution is compound:

$$u = (I - Q)u + Qu,$$

where Qu requires the solution of the deflated system

$$P\mathbf{S}_{TS}\tilde{\mathbf{u}}_{TS} = P\mathbf{b}_{TS}.$$

Solving the system $\mathbf{U}_\Phi \mathbf{u} = \mathbf{y}$

Here we have apply the inverses of the matrices $\mathbf{U}_{1,2,3}$ to a vector. This involves again the inverse of the matrix $\mathbf{K}_{uv\bar{p}}$, so the execution of this step is not cheap, and the price is fixed.

4.2 Setup

Tolerance for the 'outer' solver `FGMRES` is 10^{-9} . We test subdomain deflation on the Schur problem as described above with the dataset `small_weak`.

We consider two preconditioners for the Schur problem: LU and ILU decompositions for \mathbf{A}_{TS} . For the ILU we take `droptol` = 10^{-3} . The used tolerance for 'inner solver' on the Schur problem (`DGMRES`, custom) is 10^{-3} . Recall that the 'effective' tolerance for the custom method is smaller, the reason why it needs more iterations (section 2.4).

Subdomains	-	8	16	32	32	64	128
Configuration	-	(2, 2, 2)	(2, 2, 4)	(2, 2, 8)	(4, 4, 2)	(4, 4, 4)	(4, 4, 8)
Custom	154	165		155		153	
DGMRES	154	180 (7)		169 (7)		164 (7)	

Table 4.1: Inner iterations. Using subdomain deflation and a LU.

Subdomains	-	8	16	32	32	64	128
Configuration	-	(2, 2, 2)	(2, 2, 4)	(2, 2, 8)	(4, 4, 2)	(4, 4, 4)	(4, 4, 8)
Custom	240	210	210	192	260	183	168
DGMRES	240	228 (7)	259 (8)	240 (8)	267 (8)	227 (8)	208 (8)

Table 4.2: Inner iterations. Using subdomain deflation and an ILU.

Our choice of subdomain deflation vectors is similar like in the previous chapter. The only difference is, now there are no components other than temperature and salinity.

For each subdomain we create 2 deflation vectors, say z_T and z_S . They have respectively the temperature (T) component or the salinity (S) component of the vector equal to ‘1’. Together, these $2nmp$ vectors form/span the deflation space Z .

4.3 Results

First the results taking the LU decomposition of \mathbf{A}_{TS} . These can be found in table 4.1. We tested various configurations to investigate the behaviour of the method. The columns containing ‘-’ represent the solver without deflation. The number of outer (FGMRES) iterations is 6 in each test, unless stated otherwise in brackets. The displayed value is always the *total* number of iterations of the GMRES (resp. DGMRES) method.

We observe that both routines do not work. We suspect the troublemaker is LU, which works a bit too well as preconditioner. It seems nothing can beat that in a normal setup.

Now we test the same configurations using the ILU decomposition of \mathbf{A}_{TS} , table 4.2. The number of outer iterations is 6, unless stated otherwise.

The custom routine works now fairly well. With 64 subdomains it decreases the number of iterations by 24%. The DGMRES method on the other hand, is still performing badly. It even takes 2 more outer iterations to complete. Probably the structure of the DGMRES program is not fit for working with the Schur matrix, but we did not further investigate this.

4.4 About runtimes

The runtime of the solver is in practice more important than the number of iterations. By solving the Schur problem accurately, we only need to perform 6 outer iterations. Unfortunately, the amount of work per iterations is much larger. We will now measure the runtime of our solver and compare it to the existing solvers. These runtimes were calculated on an AMD Athlon XP 2800+ CPU and MATLAB R14.

Method	Outer iterations	Inner iterations	relres	runtime (solver only)
BGS	45	458	7.9e-10	109.1 s
BILU	6	240	4.5e-10	467.3 s
BILU-D custom	6	183	5.5e-10	336.7 s
BILU-D dgmres	8	227	2.3e-12	785.3 s
BGS-D custom	45	316	7.2e-10	99.3 s
BGS-D dgmres	45	238	7.7e-10	92.2 s

Table 4.3: Overview of runtimes for some configurations.

We work with dataset `small_weak` and an ILU decomposition of \mathbf{A}_{TS} ($\text{droptol} = 10^{-3}$). Both BILU and BILU-D solve the Schur system with tolerance 10^{-3} . BGS and BGS-D solve the simplified Schur system with the same tolerance. For the deflation solvers, we take 64 subdomains, in the configuration (4, 4, 4).

An overview of runtimes is displayed in table 4.3. We see that the number of outer iterations goes down from 45 to 6 when we use the BILU solver. BUT: the total runtime more than quadruples! In practice, the BILU solver is no match for BGS. But did we succeed in speeding it up? The answer is yes. We see that the runtime for the custom BILU-D method amounts less than 3/4 of the BILU method. But still it cannot compete with BGS.

In the following chapter we will deal with deflation on the simplified Schur problem. There we will construct the BGS-D preconditioner, also listed in the table. We see that this preconditioner can cut runtime by more than 15%.

Remark. The table only lists the runtime of the solver. When implementing deflation, some more things need to be done. For example, constructing the deflation vectors and computing recurring matrix products. We did not include the time for these things in the runtime to make a pure comparison between solvers.

Chapter 5

Deflation on the simplified Schur problem

In this chapter, we will consider deflation on the *simplified* Schur problem. That is, deflation on the system

$$\mathbf{A}_{TS} \cdot \mathbf{u}_{TS} = \mathbf{b}_{TS}, \quad (5.1)$$

where \mathbf{A}_{TS} is the Schur complement, putting the term \mathbf{B}_{TS} to zero (cf. equation (1.3)). Without deflation, this procedure gives rise to the BGS preconditioner, similar to the one currently in use in the THCM model. Thus demonstrating that deflation works on this system, could imply a speedup of the ‘real’ model. We will refer to the preconditioner *with* deflation as BGS-D.

5.1 Eigenvalues

We consider both subdomain and eigenvector deflation. For the latter we need to approximate eigenvectors of \mathbf{A}_{TS} , which is an costly business, but not priceless.

Using dataset `small_weak`, we computed the 50 smallest eigenvalues and their corresponding eigenvectors. A plot of the eigenvalues can be seen in figure 5.1. Again we see several conjugated eigenvalue pairs.

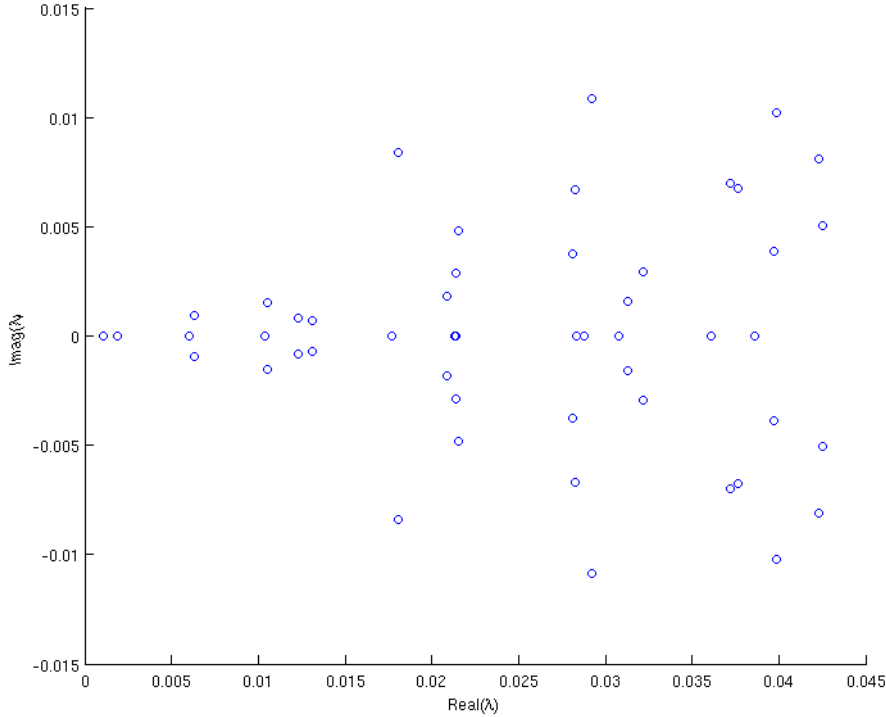
We remark that it is probably useful to apply deflation here, since there is a small number of extreme eigenvalues. Removing those should imply a significant better condition number.

5.2 Setup and results

Dataset 1: [`small_weak`] As solver for the factored system (4.1), we will turn again to FGMRES with tolerance 10^{-9} . As preconditioner for solving the \mathbf{A}_{TS} system, we use an incomplete LU decomposition (`droptol = 10^{-3}`).

The used tolerance for ‘inner solver’ on the Schur problem (DGMRES, custom) is 10^{-9} . Recall that the ‘effective’ tolerance for the custom method is smaller, the reason why it makes more iterations (section 2.4).

Our choice of subdomain deflation vectors is exactly like in the previous chapter. For each subdomain we create 2 deflation vectors, say z_T and z_S . They have respectively the temperature (T) component or the salinity (S) component of the vector equal to ‘1’. Together, these

Figure 5.1: Plot of the 50 smallest eigenvalues of \mathbf{A}_{TS} .

Eigenvectors	-	1	2	3	4	5
Custom	913	815	741	711	674	630
DGMRES	913	758	681	655	623	583

Table 5.1: Inner iterations. Using eigenvector deflation and dataset 1.

$2nmp$ vectors form/span the deflation space Z .

We start with eigenvector deflation. In table 5.1 these results can be found. The columns containing '-' represent the solver without deflation. The number of outer (FGMRES) iterations is 45 in each test. The values listed in the table are the *total* number of inner iterations.

We observe that computing 5 eigenvectors helps us decrease the number of inner iterations by 36%. This is in line with the eigenvalue plot, where we saw that there exist extremal eigenvalues. In this case we removed 5 of them.

Now we turn to subdomain deflation. The results of this are displayed in table 5.2.

Subdomains	-	2	4	8	8	16	32	32	64
Configuration	-	(1, 1, 2)	(2, 2, 1)	(2, 2, 2)	(1, 1, 8)	(1, 1, 16)	(4, 4, 2)	(2, 2, 8)	(4, 4, 4)
Custom	913	877	909	848	818	816	819	718	719
DGMRES	913	823	865	787	771	768	775	678	672

Table 5.2: Inner iterations. Using subdomain deflation and dataset 1.

Eigenvectors	-	1	2	3	4	5
Custom	1203	873^{-1}	811^{-2}	803^{-2}	782^{-2}	739^{-2}
DGMRES	1203	667^{-1}	645^{-2}	635^{-2}	621^{-2}	583^{-2}

Table 5.3: Inner iterations. Using eigenvector deflation and dataset 2.

Subdomains	-	2	4	8	8	16	32	64
Configuration	-	(1, 1, 2)	(2, 2, 1)	(2, 2, 2)	(1, 1, 8)	(1, 1, 16)	(4, 4, 2)	(4, 4, 4)
Custom	1203	1077^{-1}	1181^{-1}	1044^{-2}	968	1024^{-2}	1002^{-2}	853^{-2}
DGMRES	1203	871^{-1}	937	877^{-1}	792^{-1}	869^{-2}	827^{-1}	737

Table 5.4: Inner iterations. Using subdomain deflation and dataset 2.

The number of outer iterations is again 45 in each test. We test various configurations to investigate the behaviour of the method.

By computing vectors for 64 subdomains, we can obtain an 27% decrease in iterations. When we compare with the results for eigenvector deflation, we see this roughly corresponds to the removal of 2 eigenvalues.

Dataset 2: [small_strong] In this dataset the *forcing parameter* γ is set to ‘1’, a more realistic value than ‘0.1’ used in dataset 1. The grid size remains 16x16x16. The setup is also the same as with dataset 1.

In table 5.3 we find the results of eigenvector deflation. The number of outer iterations is 123 in each test, corrected with the number in superscript. By computing 5 eigenvectors, we can decrease the number of inner iterations by 52%. But even more surprising, with only 1 eigenvector the iterations decrease by 45%! This suggests that the extremal eigenvalues of \mathbf{A}_{TS} in this dataset are even more extreme than before.

The number of outer iterations also decreases by 2. This might be caused by the different stopping criteria for GMRES and our methods, see section 2.4.

In table 5.4 we find the results of subdomain deflation. Constructing 128 vectors on 64 subdomains cuts out 39% of the iterations. This percentage is not as high as the one found with eigenvector deflation. This could perhaps be explained by the slightly more complex nature of the eigenvectors here. In figure 5.2 we see two cross sections. The eigenvectors correspond to the smallest eigenvalue in dataset respectively dataset 2. We see that the right eigenvector is higher frequent, making it harder to approximate with subdomain vectors.

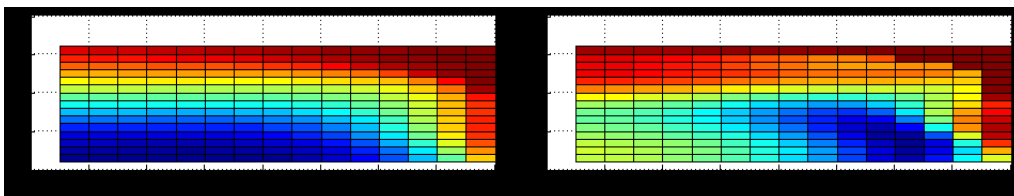


Figure 5.2: Left: cross section of eigenvector from dataset 1. Right: cross section of eigenvector from dataset 2.

Eigenvectors	-	1	2	3	4	5
Custom	35	32	29	28	27	25
DGMRES	35	29	26	25	24	22

Table 5.5: Iterations on one simplified Schur system. Using eigenvector deflation and dataset 3.

Dataset 3: [big_strong] The forcing parameter γ in this dataset is ‘1’ as well. The grid size is also larger now: $32 \times 32 \times 16$. With this dataset we run into problems. When we want to solve the entire system using FGMRES like before, our PC (with 1792MB RAM) runs out of memory. To get around this, we will not solve the entire system anymore. Instead we solve just 1 time the system (5.1).

The setup is as follows. As preconditioner for the \mathbf{A}_{TS} system, we use an incomplete LU decomposition ($\text{droptol} = 10^{-3}$). The tolerance of the solver is 10^{-9} . However, both methods finish with residues way below this tolerance. In the 512-subdomain case, the dgmres solver finishes with $\text{relres} = 7.34901\text{e-}12$, whereas the custom solver finishes with $\text{relres} = 2.20511\text{e-}12$. Note that the relres for the custom solver is 3 times smaller than the relres for dgmres. We expected that this would happen in section 2.4, where we talked about stopping criteria.

In table 5.5 we see the number of iterations for our solvers on this system. Eigenvector deflation is used here. We see that computing 5 eigenvectors cuts iterations by 37%.

In table 5.6 we see the number of iterations with subdomain deflation. Since the grid size is larger in this dataset, we also consider more subdomain configurations. It is good to see that subdomain deflation still works very well on this dataset. Subdomain deflation with 512 subdomains cuts iterations by 54%!

Subdomains	-	2	4	4	8	8	16	32
Configuration	-	(1, 1, 2)	(2, 2, 1)	(1, 1, 4)	(2, 2, 2)	(1, 1, 8)	(1, 1, 16)	(2, 2, 8)
Custom	35	31	33	29	29	29	29	25
DGMRES	35	28	31	27	27	27	27	24

Subdomains	64	128	256	256	512
Configuration	(4, 4, 4)	(4, 4, 8)	(8, 8, 4)	(4, 4, 16)	(8, 8, 8)
Custom	24	21	21	20	17
DGMRES	22	19	20	19	16

Table 5.6: Iterations on one simplified Schur system. Using subdomain deflation and dataset 3.

Chapter 6

Conclusions

In the Introduction we raised the question: is it possible to construct a new preconditioner for the THCM model, based on BILU and fit out with deflation, that is faster than BGS? As we have seen in chapter 4, this is not the case.

6.1 Deflation

But we did prove that deflation works. And with the right configuration, one can say it works quite well. For example, eigenvector deflation is already a notable improvement using only a single eigenvector. This can be explained by the favorable location of the smallest eigenvalues, which we have seen in the plots.

Subdomain deflation works just as well. It is important though to choose enough subdomains. Furthermore, it seems that increasing subdomains in the vertical direction has more effect than in the horizontal. This is of course dependent on the equations that we work with. Since we applied deflation mainly on the (simplified) Schur system, the reason for this a-symmetry can be found there. The dominating frequencies in the vertical direction are probably higher frequent than in the horizontal direction.

We have seen in section 4.4 that the runtime of the BGS preconditioner can be reduced. However, we did not take the ‘extra’ costs in account which come with deflation.

We recommend further research on deflation, as it is a very promising technique. So far, deflation works well with the linear systems occurring in THCM. However, there are still a couple of things unclear which should be investigated. We included these recommendations in the following section.

6.2 Recommendations for further study

Before implementing deflation, or disregarding it, one should consider the following research topics.

- Do the results scale to the real model? And how about the extra costs that come with deflation?
- The LU and ILU decompositions used in our tests are rather artificial. It is of great importance to research performance with the available preconditioners for \mathbf{A}_{TS} .
- As Dopheide already mentioned in his MSc thesis, the restart value is important. Making restarts in the solver might render deflation more valuable.
- Also of interest is the configuration of the subdomains. Since we worked mostly on a small grid (16x16x16), not many configurations could be tested.
- Perhaps a combination between BGS and BILU is possible. One could for example solve the Schur complement in the first outer iteration, but not in the next. With GMRES minimizing property, this could speed up the BGS solver.

Dankwoord

Graag wil ik hierbij de mensen bedanken die mij geholpen hebben tijdens de uitvoering van mijn bachelor onderzoek.

Allereerste natuurlijk dr.ir. Fred Wubs, mijn eerste begeleider, die op korte termijn deze opdracht voor mij uitgekozen heeft. Het bleek een schot in de roos, want ik vond het een erg uitdagend onderwerp. Tijdens onze voortgangsgesprekken kwamen dikwijls goede ideeën boven tafel, waar ik zelf nog niet aan had gedacht.

Ten tweede wil ik Jonas bedanken voor alle hulp, uitleg en programma's die hij mij gegeven heeft. Jij wist op elke vraag een antwoord, dat vervolgens ook altijd juist bleek te zijn. Zonder jouw kennis van het model en van MATLAB was ik niet ver gekomen. Bedankt ook voor alle aanvullingen die op mijn verslag gegeven hebt.

Als laatste wil ik Nicole bedanken voor haar oneindige vertrouwen op een goede afloop.

Bibliography

- [1] Arie de Niet, *Solving Large Linear Systems in an Implicit Thermohaline Ocean Model*, PhD dissertation, Juni 2007.
- [2] J. Frank and C. Vuik, *On the construction of deflation-based preconditioners*, SIAM J. Sci. Computing, vol 23: 442–462, 2001.
- [3] Bart Dopheide, *Experimenting with deflation-based preconditioning*, MSc thesis, University of Groningen, October 2004.
- [4] Y. Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. Sci. Computing, 14(2):461–469, 1993.
- [5] R. A. Nicolaides, *Deflation of Conjugate Gradients with Applications to Boundary Value Problems*, SIAM J. Num. Analysis, vol 24: 355–365, 1987.
- [6] Circulación termohalina, Wikipedia.
- [7] Fully Implicit Ocean Models, homepage Institute for Marine and Atmospheric research Utrecht (IMAU), last update: December 14, 2007.
http://www.phys.uu.nl/~wwwimau/research/ocean/nonlinearocean/ocean_models.