university of groningen

faculty of mathematics and natural sciences

Efficient Computation of Periodic Orbits in Space-Time Discretised Nonlinear Dynamical Systems

Jan Willem Greidanus

Master Thesis in Applied Mathematics
Supervisor(s): F.W. Wubs, M.E. Dür
September 1, 2010

Efficient Computation of Periodic Orbits in Space-Time Discretised Nonlinear Dynamical Systems

Jan Willem Greidanus

Supervisor(s):
F.W. Wubs, M.E. Dür
Institute of Mathematics and Computing Science
P.O. Box 407
9700 AK Groningen
The Netherlands

Summary

In many dynamical systems one can find periodic solutions, for example in ocean flow dynamics, where the seasonal cycle of the atmosphere imposes a periodic forcing. We can formulate these problems as a four dimensional nonlinear system in which time is included, in other words time stepping can be omitted. In order to solve these systems Newton's method is used. In the execution of the Newton method we have to solve sparse large linear systems, which is done by GMRES. For an efficient solve with GMRES the use of a preconditioner is essential. In this report such a preconditioning technique is presented, the advantage of this technique is that it allows for easy parallelisation. In this thesis this technique is tested for 1-D shallow-water equations and the THCM ocean model and the performance is compared with existing approaches.

Contents

Contents

Chapter 1

Introduction

An important part of mathematics nowadays is the study of (systems) of (partial) differential equations. In many cases analytic solutions are hard to find or it is even impossible to solve the equations analytically. In these cases we turn to numerical mathematics and try to find solutions with the use of numerical methods and computers. In this way we find a discrete solution on a user defined grid, in other words a discrete solution is found. One of the main issues in the study of numerical mathematics is the study of fluid dynamics and that is where the models in this thesis are from.

Often we are dealing with systems of partial differential equations with apart from space also a time dimension. These time dependent systems appear everywhere in physics. We are interested in the motions in a system and how the dynamics change in time. In many occasions a system may show periodic behaviour in time. With a corresponding four dimensional (XYZT-) nonlinear system we can find these periodic solutions in an efficient way. We solve for a periodic orbit at once instead of solving per time step.

For these 4-D systems where periodic behaviour occurs a preconditioning technique is developed at the University of Groningen. In this thesis this technique is described and two models in which we implemented and tested it. The goal of this research is to implement the new method in an ocean circulation model and investigate how well the new preconditioner works. The simulations we do are on the North Atlantic, in the future the goal is to be able to perform simulations on the global ocean circulation. First we will apply it to simpler toy model based on the shallow-water equations and test the performance in Matlab, from which we can up to some level predict how well it can do for the ocean model. The second step is to implement it in the Trilinos version of the THCM model and test it in some simulations. With the new preconditioning technique we hopefully will be able to solve the ocean equations with all forcings in the model, since with the existing methods this takes a lot of time and in some cases there is only convergence with only wind forcing, either the convergence is poor, or there is no straight forward way for parallelisation in time. This means the new preconditioning technique has to be able to take into account temperature and salinity forcing next to wind forcing and we need an algorithm which allows for easy parallelisation.

This thesis consists of 5 Chapters, this is the first. The second contains the problem description, a description of our new preconditioner and some theoretical background information. The third chapter is about the toy model and the results of the tests with this model. Chapter four consists of the description of the ocean model and the implementation of the preconditioner and describes the results of the new solver applied to the ocean model. In chapter five one finds a discussion of the results and conclusions.

Chapter 2

Problem Description

In this chapter I will describe the mathematical basis of this thesis. What we want to do is implement and test a solver that looks for periodic orbits of dynamical systems without explicitly integrating in time. This method can in principle be applied to any dynamical system with possible (quasi-) periodic orbits. Furthermore I will describe some theoretical background and existing methods to solve the time space nonlinear problem. For a more detailed description of the theory of dynamical systems and periodic orbits see [21] and [5].

2.1 Periodic Orbits in Dynamical Systems

A dynamical system can be seen as the time evolution of some physical system, in our case the main interest is in ocean flow. The dynamics are usually described by differential equations. Two important qualitative concepts in the study of dynamical systems are stationary and periodic motions. A dynamical system is in a stationary state if the state does not change in time. In case of a periodic motion or periodic orbit we have that for a solution y,

$$y(t+T) = y(t),$$

for any time t and a minimum $T > 0$ which is the period. Note that for the special case $T = 0$ we end up with a steady state solution. In general the dynamics is not periodic right away. We start in such a case with a periodic forcing and after some time the so-called limit cycle is reached. This is also important in the ocean dynamics since looking for periodic orbits often requires a long spin up time, i.e. the time after which the limit cycle is reached. This spin up problem takes a lot of computational time and therefore methods are developed to find periodic orbits quicker. This is a big advantage if one wants to perform numerical continuation. Because of the seasonal forcing, we only consider the non-autonomous case, the method can also be applied to autonomous systems, however this is more complicated.

2.2 Methods

We are interested in dynamical systems that can be written in matrix notation as:

$$M\frac{dx}{dt} = F(x), \tag{2.1}$$

where x is a vector which contains the unknowns. M is the so called mass matrix and is constant in time. F is the (nonlinear) discretised system of equations describing the dynamics. If we do a general finite difference approximation with a first order approximation of the time derivative we can write

$$M\frac{x(t+\Delta t) - x(t)}{\Delta t} = \theta F(x(t+\Delta t)) + (1-\theta)F(x(t)) \tag{2.2}$$

For $\theta = 1$ we have the Backward Euler method. This choice leads to fully implicit method, which means on the right-hand side only terms with $t+\Delta t$ appear and we have to solve a large nonlinear system of equations. (Fully) implicit methods have in comparison with explicit methods the advantage that the time step is not limited by a stability condition, so the limiting factor of the time-step is accuracy. The choice for an explicit time stepping method gives limitations on the time-step because of numerical stability conditions, such as the CFL condition. The disadvantage of implicit time-integration is that we have to solve a large and in many cases nonlinear system of equations in every time step. However we have a good solver available. We now have

$$Mx(t+\Delta t) - Mx(t) = \Delta t F(x(t+\Delta t)), \tag{2.3}$$

for all k steps in a periodic solution. Now if we call the periodic solution $x_1, x_2, x_3, \ldots, x_k$, we can write,

$$Mx_i - Mx_{i-1} = \Delta t F(x_i), \tag{2.4}$$

where $x_1 = x_{k+1}$, which means we have k time steps per periodic orbit. Rewriting gives

$$Mx_i - Mx_{i-1} - \Delta t F(x_i) = 0. \tag{2.5}$$

This means we can write the system for the whole periodic solution in the following form.

$$\begin{bmatrix} -\Delta \mathrm{tF(x_1)} + \mathrm{Mx_1} - \mathrm{Mx_k} \\ -\mathrm{Mx_1} - \Delta \mathrm{tF(x_2)} + \mathrm{Mx_2} \\ \vdots \\ -\mathrm{Mx_{k-2}} - \Delta \mathrm{tF(x_{k-1})} + \mathrm{Mx_{k-1}} \\ -\mathrm{Mx_{k-1}} - \Delta \mathrm{tF(x_k)} + \mathrm{Mx_k} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}. \tag{2.6}$$

If we want to solve this by Newton's method we have to calculate the Jacobian, which has the block structure as in equation 2.7 in this case.

$$\begin{bmatrix} -\Delta \mathrm{tA(x_1)} + \mathrm{B} & 0 & \ldots & 0 & -\mathrm{B} \\ -\mathrm{B} & -\Delta \mathrm{tA(x_2)} + \mathrm{B} & 0 & \ldots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ 0 & 0 & -\mathrm{B} & -\Delta \mathrm{tA(x_{k-1})} + \mathrm{B} & 0 \\ 0 & \cdots & 0 & -\mathrm{B} & -\Delta \mathrm{tA(x_k)} + \mathrm{B} \end{bmatrix}, \tag{2.7}$$

where A contains the derivatives of the time independent part F with respect to the variables and B is the Jacobian of the time dependent part Mx.

2.2.1 Finding Solutions

Usually one finds a solution by performing a time simulation in which equation 2.5 is solved in every time step. However we will look for a solution of a periodic orbit at once by solving the four dimensional system 2.6 for one period at once.

Since in both cases the systems of equations are nonlinear we will use Newtons method to solve them, for a description of Newton's method see [20]. This comes down to determining the Jacobian and right-hand side and solving the corresponding linear system until convergence of the method. For the time simulation method we use Newton's method in every time step, but for the four dimensional system we have to use the Newton algorithm only once, however with many more unknowns and equations. The large linear system which now appears in the Newton algorithm makes it interesting to look for ways to speed up this solve. This can be done in several ways. First we have to choose between a direct or an iterative solver. Since the Jacobian is very sparse, iterative solvers are preferable. Since it is nonsymmetric we choose preconditioned GMRES. The convergence of the GMRES can be improved a lot by using a preconditioner, there is however a lot of difference between performance of preconditioners. We want to exploit the structure of the matrix as much as possible in the choice of our preconditioner. Because of the block structure of the Jacobian we would like to take a preconditioner with the same structure, since we can use Fast Fourier Transformation (FFT), as described in [22] and [4] in order to diagonalise it, see [7]. In our case we want to solve the following linear system in the i-th Newton step:

$$\begin{bmatrix} -\Delta t A(x_1)+B & 0 & \dots & 0 & -B \\ -B & -\Delta t A(x_2)+B & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ 0 & 0 & -B & -\Delta t A(x_{k-1})+B & 0 \\ 0 & \cdots & 0 & -B & -\Delta t A(x_k)+B \end{bmatrix} \begin{bmatrix} dx_1^i \\ dx_2^i \\ \vdots \\ dx_{k-1}^i \\ dx_k^i \end{bmatrix}$$

$$= \begin{bmatrix} -\Delta t F(x_1)+Mx_1-Mx_k \\ -Mx_1-\Delta t F(x_2)+Mx_2 \\ \vdots \\ -Mx_{k-2}-\Delta t F(x_{k-1})+Mx_{k-1} \\ -Mx_{k-1}-\Delta t F(x_k)+Mx_k \end{bmatrix}. \tag{2.8}$$

The goal is now to construct a preconditioner to this system that exploits the structure of the matrix.

2.3 Preconditioning

2.3.1 Preconditioning for Large Linear Systems

In general there are two ways to solve large linear systems, direct or iterative. In the case of a direct solver the amount of work depends on the method, shape, pivoting and the dimension of the matrix. For iterative solvers the fill and values of the matrix are very important. With preconditioning a linear system can be transformed such that an iterative method converges faster. The idea of preconditioning is to simplify the system that has to be solved by multiplying it with a matrix that is as close as possible to the inverse of the system matrix. In formula we can write

$$AP^{-1}Px = b,$$

where P is the preconditioner. We can solve the preconditioned system via two solves. The first is via a Krylov subspace method we solve

$$(AP^{-1})y = b$$

and after that x follows from

$$x = P^{-1}y.$$

Applying the inverse of P is done by programming a function which computes the effect of the preconditioner on a vector. In fact we are solving a system with P as system matrix. We can say that the goal of the preconditioner is to make the condition number of the preconditioned matrix (AP^{-1}) as close as possible to one. The condition number of a matrix A is defined by

$$k(A) = ||A|| \cdot ||A^{-1}||,$$

for any consistent norm. A lower bound which is sharp for normal matrices in the 2-norm is given by

$$\frac{\max(abs(\lambda|\lambda \text{ eigenvalue of } A))}{\min(abs(\lambda|\lambda \text{ eigenvalue of } A))}.$$

So we want to bring the eigenvalues of A as close to one as possible. Notice that if we would take A^{-1} as a preconditioner that would give us the identity matrix as preconditioned matrix, so then all eigenvalues are one. A choice could be to take the diagonal of the matrix as a preconditioner, this is very easy to apply, however we can do a lot better. The choice of what preconditioner to use is model dependent. In our case the matrix that is to be preconditioned is the Jacobian. The block structure of the matrix in our problems makes it interesting to apply preconditioners which inherit this block structure, because this makes them easy and thus quick to construct, a more detailed description of this block structure can be found in the following sections. The preconditioner we develop has the characteristic that it allows for easy parallelisation. This is mandatory as the 4-D system becomes very large in real world applications.

2.3.2 Fast Fourier Transform

In the Jacobian (2.7) we replace in the diagonal blocks $A_i, i = 1, \ldots, k$ by $A = A(\bar{x})$, where $\bar{x}$ is the mean of the k solutions. The obtained matrix

$$W := \begin{bmatrix} -\Delta tA & 0 & \ldots & 0 & -\mathrm{B} \\ -\mathrm{B} & -\Delta tA & 0 & \ldots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ 0 & 0 & -\mathrm{B} & -\Delta tA & 0 \\ 0 & \cdots & 0 & -\mathrm{B} & -\Delta tA \end{bmatrix}, \tag{2.9}$$

can be used as a preconditioner for (2.8). The only difference with the actual Jacobian is that we used the mean solution in the A blocks, the structure is completely the same. In order to fully use this advantage we should be able to apply parallelisation in order to speed up the solves, how this can be done is described below. We apply a Fourier transform to the linear system, the Fourier transform is then the coupling between the k systems.

The matrix (2.9) is a block circulant matrix and can be written in Kronecker products, so

$$W = -I_k \otimes \Delta tA + C_k \otimes B, \tag{2.10}$$

where I_k is the $k \times k$ identity matrix and

$$C_k = \begin{bmatrix} 1 & 0 & \dots & 0 & -1 \\ -1 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ 0 & 0 & -1 & 1 & 0 \\ 0 & \cdots & 0 & -1 & 1 \end{bmatrix} \in \mathbb{R}^{k \times k}. \tag{2.11}$$

As mentioned in [7] we can diagonalise C_k using an FFT (Fast Fourier Transform). The Fast Fourier Transform is an efficient algorithm to apply discrete Fourier transforms. Discrete Fourier transforms decompose sequences of values into components of different frequencies [4]. We have

$$C = F\hat{C}F^*. \tag{2.12}$$

F is the Fourier matrix, a $k \times k$ unitary matrix of the eigenvectors of C and F^* it's conjugate transpose. If we apply the FFT to the matrix system with W, we get

$$(F^* \otimes I_k)W(F \otimes I_k)(F^* \otimes I_k)u = (F^* \otimes I_k)f. \tag{2.13}$$

Now we apply 2.10,

$$(F^* \otimes I_k)W(F \otimes I_k) = (F^* \otimes I_k)(-\Delta t I_k \otimes A + C_k \otimes B)(F \otimes I_k) = -\Delta t F^*F \otimes A + F^*CF \otimes B = \tag{2.14}$$

$$-\Delta t I_k \otimes A + \hat{C} \otimes B. \tag{2.15}$$

Here $\hat{C}_k$ is a complex $k \times k$ diagonal matrix. Since C is a banded circulant matrix the eigenvalues are as given in [7]:

$$\lambda_j = c_0 + [c_1 + c_{n-1}] \cos\left(\frac{(j-1)2\pi}{k}\right) + i[c_1 - c_{n-1}] \sin\left(\frac{(j-1)2\pi}{k}\right), \tag{2.16}$$

for $j = 1, \dots, k$. c_j Is the j-th element of the first row of the circulant matrix, in our case $c_1 = 0, c_0 = 1$ and $c_{n-1} = -1$, this means we have

$$\lambda_j = 1 - e^{\frac{ij2\pi}{k}}. \tag{2.17}$$

If we look at the eigenspectrum we find the circular pattern of figure 2.1. Hence the eigenvalues are all 'extreme'. GMRES applied to a system with such a matrix will stagnate for the first $k-1$ steps and converge at the k-th step.

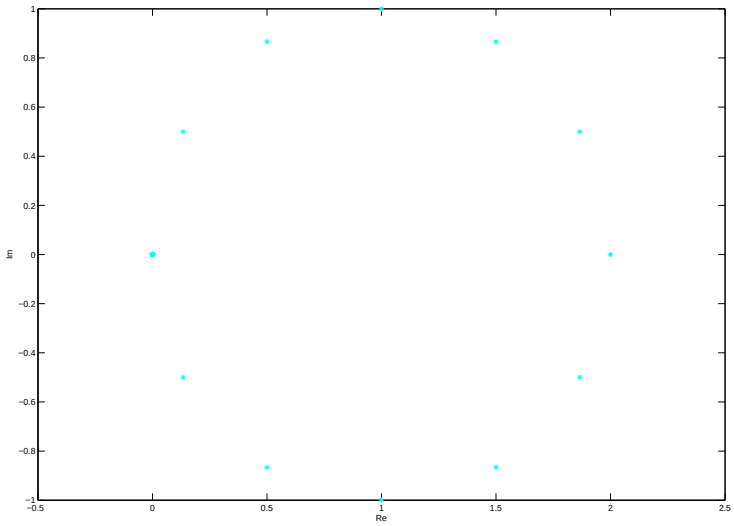


Figure 2.1: Eigenspectrum for circulant matrix 2.11

Now we have a block diagonal matrix with the following diagonal elements

$$\hat{W}_j = -\Delta t A + \lambda_j B = -\Delta t A + B - e^{\frac{ij2\pi}{k}} B, j = 1, \ldots, k. \tag{2.18}$$

In order to solve the system with the preconditioner we now also have to do inverse FFT on the right hand side f, in equation 2.13. We have to do this for every unknown separately, which means we do m fast Fourier transforms on vectors of length k, this is

$$f = \begin{bmatrix} f^{(1)} \\ f^{(2)} \\ \vdots \\ f^{(k)} \end{bmatrix} \tag{2.19}$$

where $f^{(j)}$ is a column vector with length m. We now apply the FFT to the vectors consisting of the i-th elements of $f^{(j)}, \forall j$, $i = 1, \ldots, m$. And after solving the system we have to transform the found solution back using FFT's in the same way. This process is repeated in every GMRES-iteration. Although we apply the Fourier transforms to a real valued system the application of a Fourier transform in general returns a complex linear system. This gives some additional difficulties that need to be solved, this is described in section 2.5.

2.4 Already Available (Trilinos) Preconditioners

In the Trilinos package LOCA there are already some preconditioners for solving space-time (XYZT) systems available. These are described below and we compared their performance later in this thesis.

2.4.1 Block Diagonal (Block-Jacobi)

The first preconditioner we look at is the Block-Jacobi preconditioner. This has the following form

$$\begin{bmatrix} -\Delta tA(x_1)+B & 0 & \dots & 0 & 0 \\ 0 & -\Delta tA(x_2)+B & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ 0 & 0 & 0 & -\Delta tA(x_{k-1})+B & 0 \\ 0 & \cdots & 0 & 0 & -\Delta tA(x_k)+B \end{bmatrix}, \tag{2.20}$$

it is the Jacobian matrix without the off diagonal blocks. This preconditioning technique is easy to implement in parallel as all blocks can be solved independently. However we do not expect it to scale well when the number of time-steps increases: the more time-steps, the more subdiagonal blocks are dropped, which gives a worse approximation of the Jacobian (2.7). This has influence on the performance of the preconditioner, since the eigenvalues of the preconditioned system are not scaled very close to one.

2.4.2 Sequential

The Sequential preconditioner is a non parallel preconditioner. This is a disadvantage especially for large scale real world applications. The idea is that the upper right block in the matrix (2.7) is omitted in the preconditioner. If we do this we get a block lower triangular matrix as preconditioner. Solving a system with this matrix comes down to repeatedly solving a system with a diagonal block as system matrix, for the latter we have a good solution method. The preconditioner thus looks like:

$$\begin{bmatrix} -\Delta tA(x_1)+B & 0 & \dots & 0 & 0 \\ -B & -\Delta tA(x_2)+B & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ 0 & 0 & -B & -\Delta tA(x_{k-1})+B & 0 \\ 0 & \cdots & 0 & -B & -\Delta tA(x_k)+B \end{bmatrix}. \tag{2.21}$$

The Parallel and ParaReal preconditioners are (parallel) improvements to the sequential preconditioner.

2.4.3 Parallel

The parallel preconditioner comes down to next to the upper right block also neglecting some of the blocks below the diagonal blocks, which opens the door to parallelisation. This preconditioner simultaneously applies the sequential algorithm across the decoupled time domains. This means there is no communication of solutions between time domains. The separate time domains can this way be computed on separate processors. We have to keep in mind that the coupling between the time domains remains important, so we cannot compute every time step on a different processor. Also a too small number of time steps on a time domain might cause slow convergence. For example if we choose three time steps per time domain with a total of six the preconditioner looks like

$$\begin{bmatrix} -\Delta\mathrm{tA}(\mathrm{x}_1)+\mathrm{B} & 0 & \dots & 0 & 0 & 0 \\ -\mathrm{B} & -\Delta\mathrm{tA}(\mathrm{x}_2)+\mathrm{B} & 0 & \dots & 0 & 0 \\ 0 & -\mathrm{B} & -\Delta\mathrm{tA}(\mathrm{x}_3)+\mathrm{B} & 0 & \dots & 0 \\ 0 & 0 & 0 & -\Delta\mathrm{tA}(\mathrm{x}_4)+\mathrm{B} & 0 & 0 \\ 0 & \dots & 0 & -\mathrm{B} & -\Delta\mathrm{tA}(\mathrm{x}_5)+\mathrm{B} & 0 \\ 0 & 0 & \cdots & 0 & -\mathrm{B} & -\Delta\mathrm{tA}(\mathrm{x}_6)+\mathrm{B} \end{bmatrix}. \tag{2.22}$$

For one time step per time domain the Parareal Solver is equal to block Jacobi.

2.4.4 Parareal

In the parareal preconditioner the communication between separate time domains which was dropped in the Parallel case is to some level restored. The time domains are the same as in the Parallel algorithm. However also a sequential preconditioning is applied on the first time steps of every time domain. The advantage in comparison with Parallel preconditioning is that an estimate of the solution from the time step on the previous domain is computed to help accelerate the convergence. This estimate is applied to the B blocks that where dropped in the parallel case. There are several ways this can be implemented. For more information on the Parareal in Time solver see [23].

2.5 Solving Complex-Valued Linear Systems

In order to solve the complex system, we will reformulate it as an equivalent real valued system as described in [8]. If we do this we do not have to use complex valued variables within Trilinos. The best possibility within Trilinos is usage of the KomplexLinearProblem class. This solves an equivalent real linear system in order to avoid the usage of complex numbers. This means we can use general methods for real valued linear systems. We can write the general complex system in k separate systems as

$$[\Delta tA + (1-\cos\left(\frac{j2\pi}{k}\right))B - i\sin\left(\frac{j2\pi}{k}\right)B](x^j + iy^j) = (f_r^j + if_i^j), \tag{2.23}$$

or equivalently

$$[\Delta tA+(1-\cos\left(\frac{j2\pi}{k}\right))B]x^{(j)}+[\Delta tA+(1-\cos\left(\frac{j2\pi}{k}\right))B]iy^{(j)}-[i\sin\left(\frac{j2\pi}{k}\right)B]x^{(j)}+[\sin\left(\frac{j2\pi}{k}\right)B]y^{(j)} = (f_r^j + if_i^j). \tag{2.24}$$

Here the subscripts r and i stand for real and imaginary part respectively and $i = \sqrt{-1}$. Now we have two separate systems, one real valued and one pure imaginary, so

$$[\Delta tA + (1-\cos\left(\frac{j2\pi}{k}\right))B]x^{(j)} + [\sin\left(\frac{j2\pi}{k}\right)B]y^{(j)}) = f_r^j, \tag{2.25}$$

$$[\Delta tA + (1-\cos\left(\frac{j2\pi}{k}\right))B]y^{(j)} - [\sin\left(\frac{j2\pi}{k}\right)B]x^{(j)} = f_i^j. \tag{2.26}$$

Which gives us an equivalent real valued system, with double the amount of unknowns in comparison with the original system. If we take

$$E := [\Delta tA + (1-\cos\left(\frac{j2\pi}{k}\right))B]$$

and

$$F := [\sin\left(\frac{j2\pi}{k}\right) B],$$

we can write

$$\begin{bmatrix} E & F \\ -F & E \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f_r \\ f_i \end{bmatrix}.$$

Now we have a real valued system, with a solution consisting of x, which is the actual real part and y which is the actual imaginary part. The complete solution has to be Fourier transformed as described in section 2.3.2.

Chapter 3

Shallow-Water Model

In order to test the application of the new preconditioning method we will start using it on a toy model based on the shallow water equations. So we can see how it works in a simple setting. We will look at the one dimensional shallow water equations as described in [3]. These equations describe the waterheight and velocity on a 1-D grid. The domain will be in the same proportions that occur in ocean models, the domain is bounded by $x = 0$ on the left side and $x = L$ on the right side. We will use a change of variables in order to get a constant mass matrix (see 2.1), to keep the similarity with the later application in the ocean model intact. The mass matrix has become non-constant because of the discretisation technique used, this discretisation technique is applied because of the robustness properties. We want to solve the time dependent shallow water equations. The boundary conditions are given by zero velocity at the left of the domain and a time-periodic depth around the equilibrium depth at the right side of the domain. To solve the equations we discretise them in space and to perform time-stepping we introduce the implicit Euler method for the time derivatives. This procedure gives a non-linear system of equations for every time step. The straightforward way to solve such a problem would be to apply Newton's method in every time step. However, this involves a lot of linear systems to be solved. We are interested in periodic solutions for a system of equations, this means we can resolve the dynamics within a finite number of time-steps. We will combine the nonlinear systems for all time-steps into a single nonlinear system for all time steps. Thus we effectively solve for a steady state of the space-time discretised equations in a single Newton process. The system we want to solve has dimensions of number of time steps in a period (k) times dimension of a system for one time-step (n).

3.1 Equations

The equations governing the shallow water dynamics are given by

$$(du)_t + (du^2)_x + gd\eta_x = 0 \tag{3.1}$$

and

$$d_t + (du)_x = 0. \tag{3.2}$$

Where $d(x,t)$ is the depth and $u(x,t)$ is the velocity. The displacement from the mean level is $\eta = d - d_0$ In order to find solutions we need to prescribe two boundary conditions, these will be given by

$$u(0,t) = 0 \tag{3.3}$$

and

$$d(L,t) = d_0 + \frac{d_0}{20}\sin\left(\frac{2\pi t}{3600\cdot 24\cdot 365}\right). \tag{3.4}$$

Here d_0 is the equilibrium depth. By varying the depth on the right side of the domain we add a forcing to the dynamical system and the period of the forcing is chosen to be one year.

If we write the equations in $v = ud$ and d instead of u and d we get

$$(v)_t + \left(\frac{v^2}{d}\right)_x + gd\eta_x = 0 \tag{3.5}$$

$$d_t + (v)_x = 0. \tag{3.6}$$

Since d_0 is independent of x and using the chain rule we can write the first equation as

$$(v)_t + \left(\frac{v^2}{d}\right)_x + \left(\frac{1}{2}gd^2\right)_x = 0. \tag{3.7}$$

This different choice of variables is useful, because when using u we get a much more complicated Jacobian, especially in the time derivative part, which would then depend on u and d, whereas after the substitution this is not the case.

3.2 Discretisation

The equations are discretised in the middle of two successive grid points. This has advantages for robustness of the method. After discretisation we get the following for the time independent part.

$$f_{i+1} = \frac{v_{i+1}^2}{d_{i+1}} + \frac{1}{2}gd_{i+1}^2 - \frac{v_i^2}{d_i} - \frac{1}{2}gd_i^2 + \frac{1}{2}gd_{i+1}(h_x)_{i+1} + \frac{1}{2}gd_i(h_x)_i\Delta x \tag{3.8}$$

$$f_{i+n} = v_{i+1} - v_i \tag{3.9}$$

From this we can calculate the Jacobian, since equation $i+1, i = 1,\ldots,n-1$ depends on $v_i, v_{i+1}, d_i, d_{i+1}$ we only have the following nonzero elements

$$\frac{\partial f_{i+1}}{\partial v_i} = -\frac{2v_i}{d_i} \tag{3.10}$$

$$\frac{\partial f_{i+1}}{\partial v_{i+1}} = \frac{2v_{i+1}}{d_{i+1}} \tag{3.11}$$

$$\frac{\partial f_{i+1}}{\partial d_i} = \frac{v_i^2}{d_i^2} - gd_i + \frac{g}{2}(h_x)_i\Delta x \tag{3.12}$$

$$\frac{\partial f_{i+1}}{\partial d_{i+1}} = -\frac{v_{i+1}^2}{d_{i+1}^2} + gd_{i+1} + \frac{g}{2}(h_x)_{i+1}\Delta x \tag{3.13}$$

for equation $i+n,\ldots,i+2\times n-1$ we have

$$\frac{\partial f_{i+n}}{\partial v_i} = -1 \tag{3.14}$$

$$\frac{\partial f_{i+n}}{\partial v_{i+1}} = 1 \tag{3.15}$$

Since they only depend on v_i, v_{i+1}.

For the time dependent part we now have

$$\begin{bmatrix} v \\ d \end{bmatrix}_t \tag{3.16}$$

if we want to use this in between the grid points again we have to apply a averaging operator. This can be done by pre-multiplying with a matrix, we call it M. This matrix is defined below

$$M := \begin{bmatrix} 0 & 0 & \dots & \dots & \dots & \dots & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & \dots & \dots & \dots & \dots & 0 \\ \vdots & \ddots & \ddots & & & & & \vdots \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & \dots & \dots & 0 \\ 0 & \dots & \dots & 0 & \frac{1}{2} & \frac{1}{2} & 0\dots & 0 \\ 0 & \dots & \dots & 0 & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & & & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \dots & \dots & \dots & \dots & 0 & 0 \end{bmatrix}. \tag{3.17}$$

At this point we can write the discretised equations in matrix vector form

$$M \begin{bmatrix} \frac{dv}{dt} \\ \frac{dd}{dt} \end{bmatrix} = F(x), \tag{3.18}$$

which is actually a similar equation as (2.1) and therefore we can apply the proposed method of Chapter 2.

3.3 Results of the Simulations

Below one finds the results for the block Circulant preconditioner in comparison with the block Jacobi preconditioner and the case without preconditioning for the shallow water model forced by (3.4). For all experiments below we took as tolerance for the stopping criterion for GMRES 10^{-10} and for the Newton method 10^{-8}. The number of grid points is doubled a few times, in contrary the time step is halved several times in every experiment. The results are all for a flat bottom, with a domain size of $L = 10000km$ and (begin) depth d is varied. Of course doubling the number of grid points gives a system with the double number of unknowns, as well as halving the time-step does. The time steps are in weeks. One can imagine that increasing the number of time steps gives better accuracy We will also look at the generalised eigenvalue problem for the Jacobian and the preconditioner, which gives in fact the eigenvalues of the preconditioned matrix and are important for the speed of convergence of GMRES.

3.3.1 Solutions of the Model

We solved the shallow water equations described in the previous section using our space time method and also using normal time stepping. In many cases we do not get a periodic solution right away with the time stepping method. This becomes visible in figures 3.1 and 3.2, where we see a time series of the depth in center of the grid. The timeseries is a sequence of solutions for a certain unknown in time. Because of the periodicity we expect the periodic behaviour to appear in every unknown separately.

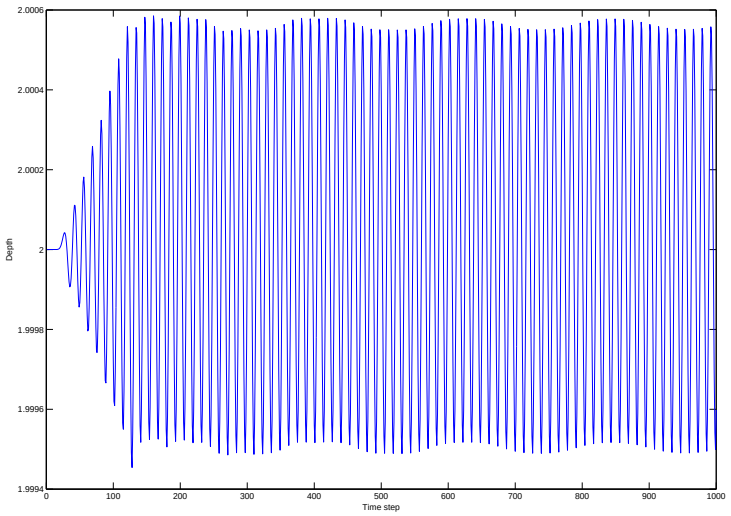


Figure 3.1: Timeseries for the shallow water equations, depth in the middle of the domain.

From the figures we see that in the beginning the solutions are clearly not periodic, but after some time the solution gets periodic more and more and becoming completely periodic in the end. In the first figure we see some irregular behaviour in the beginning, the first 600 time steps. In the second figure the solutions slowly tends to periodic after about 10000 time steps. The

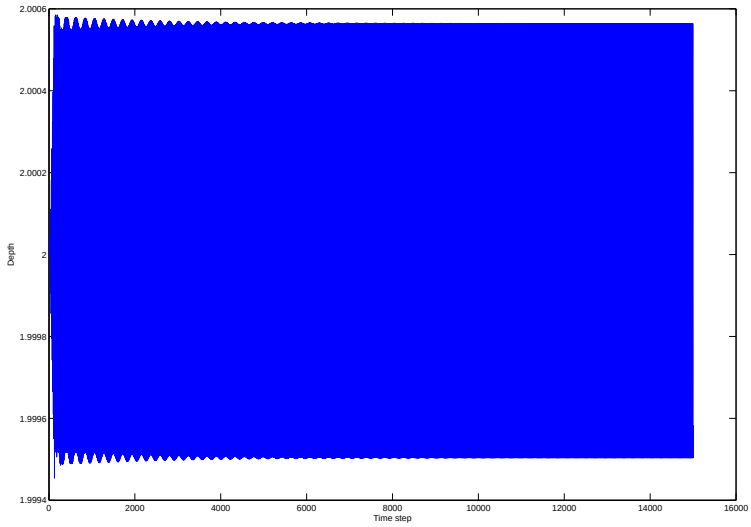


Figure 3.2: Timeseries for the shallow water equations, depth in the middle of the domain.

time before reaching periodic solutions is called the spin up time[1]. If we run the time simulation long enough we can compare the solution with the solution from the 4-D system. This gives the same results, as expected. However it is important to take enough time steps per periodic orbit in the time space method in order to find the correct periodic solutions. The solution for an unknown in one period typically looks like figure 3.3

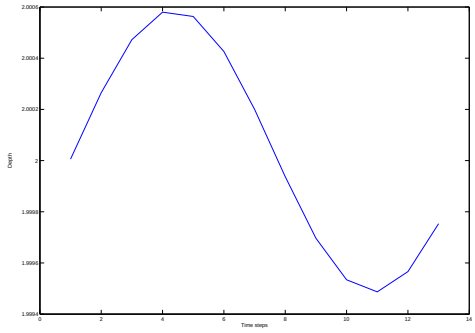


Figure 3.3: Timeseries for a periodic solution from the 4-D system, depth in the middle of the domain.

[1] Actually in this case the spin up time depends on the grid, for linearised equations we found for the continuous case purely imaginary eigenvalues and for the discrete case they were nonpositive for subcritical flows, so there is numerical dissipation in the model which decreases if the grid is refined, see [14]. Therefore it would have been better to add physical damping to the equations by adding a diffusion term ($\mu\Delta u$), or a bottomfriction term ($-\lambda u$) to the momentum equation.

3.3.2 Iterations

Below are tables with the total number of GMRES iterations and average number of GMRES iterations per Newton step, in all cases the Newton method converged in a maximum of four steps. In the tables total means the total number of GMRES iterations needed for convergence of the Newton process, av (average) is the average number of GMRES iterations per Newton step. Under methods we see either no preconditioning, which means this is the case without preconditioning, or block with respectively block Jacobi, circulant (this means the actual circulant matrix applied as preconditioner without using FFT's) or circulant fft (this is the same as Circulant but now with application of FFT's). Circulant and circulant FFT are in fact two implementations of the preconditioner in equation 2.9. Their numerical behaviour is therefore the same, but the FFT-based implementation is substantially faster. We can see from the tables that using a preconditioner gives a big advantage in comparison to solving the linear system without preconditioning.

Table 3.1: Number of iterations for $n = 10$, and $d = 2$ (km) Δt in weeks

Δt	4		2		1		0.5		0.25		0.125	
method	total	av	total	av	total	av	total	av	total	av	total	av
no preconditioning	544	136	571	143	577	144	579	145	592	148	609	152
block Jacobi	16	4	16	4	20	5	20	5	23	6	28	7
block circulant	20	5	20	5	20	5	20	5	20	5	20	5
block circulant fft	20	5	20	5	20	5	20	5	20	5	20	5

When the time step (in weeks) is made smaller, the results in table 3.1 show the same amount of iterations for the circulant preconditioner in contrast to the block Jacobi the number of GMRES iterations increases if we decrease the time step. This is due to the fact that for smaller time steps the off diagonal part of the Jacobian matrix (B) becomes larger in magnitude and because in case of a larger number of time steps more off diagonal blocks are neglected in the Block Jacobi case. These off diagonal parts are taken into account by the block circulant preconditioner.

Table 3.2: Number of Iterations for $n = 40$, and $d = 2$ (km) Δt in weeks

Δt	4		2		1		0.5	
method	total	av	total	av	total	av	total	av
no preconditioning	2203	551	2261	565	2287	572	2295	574
block Jacobi	16	4	17	4	20	5	20	5
block circulant	20	5	20	5	20	5	20	5
block circulant fft	20	5	20	5	20	5	20	5

The number of unknowns does not seem to effect the number of iterations, this can be expected because in the preconditioner only approximations in time are applied. The same effect can be seen in table 3.3, where the depth is taken much smaller, which lowers the importance of the diagonal part of the Jacobian matrix, because of the d term in the discretised equations. The performance of the block Jacobi is worse for smaller time steps, however Circulant hardly

needs extra iterations.

Table 3.3: Number of Iterations for $n = 10$, $d = 0.02$ (km) Δt in weeks

Δt	4		2		1		0.5		0.25	
method	total	av	total	av	total	av	total	av	total	av
no preconditioning	657	164	766	192	798	200	866	217		
block Jacobi	19	5	20	5	24	6	31	8	41	10
block circulant	16	4	17	4	18	5	19	5	19	5
block circulant fft	16	4	17	4	18	5	19	5	19	5

3.3.3 Run Time

In the runtimes the FFT and block Jacobi preconditioners are quite competitive (see table 3.4), although block Jacobi is a bit faster, this means it needs less time per iteration. If the time step size is decreased we see that the solution times are almost the same. The circulant preconditioner stays behind in time, this is probably due to the extra computations caused by the off diagonal blocks, so the Fourier transforms really do speed up the process. Without preconditioning the run time is much longer than with preconditioning.

Table 3.4: Solution time: for $n = 10$, $d = 0.02$ (km) Δt in weeks

Δt	4	2	1	0.5	0.25	0.125
method	time	time	time	time	time	time
no preconditioning	0.7289	1.0346	1.6221	2.9921	6.5030	16.3462
block Jacobi	0.0763	0.1529	0.3633	0.9733	2.9684	9.5685
block circulant	0.1068	0.2232	0.4944	1.2411	3.7467	11.2491
block circulant fft	0.0957	0.1742	0.3862	0.9908	2.9885	9.6271

3.3.4 Results with spin up time

I repeated the simulations with a bit different settings, with which the advantages of the Circulant method become much more clear. We took much smaller scales in comparison with the simulations before. The most important difference is that for these settings there is a spin up time, which is not the case for the results above. This is indicated by the fact that for the above results we have a periodic solution right away, even with normal time stepping in contrast to the current setting. It appears that it is important whether we are dealing with a quite simple situation, where a periodic solution is the only possibility and communication between the time steps is not important, or a situation where we have a spin up time and the communication between the time steps in the Circulant preconditioner takes care of this very well.

Table 3.5 shows that in this case for a small step we already need much more iterations for block Jacobi in comparison with our method. Another important observation is that the solution

Table 3.5: Number of Iterations for $n = 40$, and $d = 2$(m) $L = 1000$(m) Δt in seconds

Δt	4		2	
method	total	av	total	av
block Jacobi	807	269	2671	668
block circulant fft	11	4	16	4

time has a factor 10 for block Jacobi if we halve the time step, against only a factor 3 for the Circulant method. This can be seen in table 3.6.

Table 3.6: Solution Time for $n = 40$, and $d = 2$(m) $L = 1000$(m) Δt in seconds

Δt	4	2
method	time	time
block Jacobi	6.25	64.42
block circulant fft	0.35	1.13

3.3.5 Eigenvalues

The purpose of the preconditioner is to lower the condition number of the matrix, this means we want to move the eigenvalues of the matrix in the linear system as close to 1 as possible. Therefore we will look at the eigenvalues of the preconditioned matrix for a few cases. Since really computing the inverse is a quite expensive operation we will look at the equivalent generalised eigenvalue problem, which is given by

$$Jx - \lambda W x = 0. \tag{3.19}$$

Where J is the Jacobian matrix and W is the preconditioner. We can make the eigenvalue properties visible by plotting the eigenspectra, these plots can be found in Appendix . In figures A.1 - A.4 we see that the eigenspectrum of a preconditioned matrix doesn't change much in between succeeding Newton iterations. This behaviour occurs in general.

By decreasing Δt we can check if the eigenspectra can illustrate the effect that for small Δt the circulant FFT preconditioner is better then the block Jacobi, which is already visible from the results above. To do this we pre-multiply the linear system matrix with the inverse of the circulant matrix (2.9), since the FFT's are only used to speed up the application of the preconditioner. For the case of the block Jacobi preconditioner the eigenvalues lie on circles, which is comparable to the eigenvalues of the circulant matrix in section 2.3.2. This is explicable because the matrix we are dealing with here is a block matrix with a circulant block structure. The radius of the circles depends on the value of Δt. For larger time steps the radius is smaller. If the time step is decreased by a factor two, we see that the magnitude of $1 - \lambda$, where λ is an eigenvalue, on the outer circle becomes twice as large. This is also the case for the inner circles. For the circulant preconditioner however this is not the case, the distance of the eigenvalues from one doesn't change if we look at the real part. The imaginary part does increase in magnitude with a factor two, but since the imaginary part of the eigenvalues is quite small (10^{-8}), this does not affect the efficiency of the preconditioner as much as for the block Jacobi case. This is visible in figure A.5 and A.7, which are plots of the eigenvalues of the preconditioned matrix according to the results in table 3.1.

The most interesting are the matrices for which the number of GMRES iterations is larger for the block Jacobi preconditioner than for the circulant. Therefore we look at the case for $\Delta t = 0.25$ (compare to table 3.1). One can see that although the eigenvalues the furthest from one are in the eigenspectrum plot of the circulant preconditioner but it does need less iterations then block Jacobi in this case. This is probably because of the structure of the eigenspectrum, since in the block Jacobi case there are many eigenvalues with the same distance to one as for the circulant case there are only two furthest away, see figure A.9. For the case with $d = 0.02$ we can see much clearer why the circulant preconditioner works better, the magnitude of $1 - \lambda$ hasn't changed for the circulant preconditioned system. For the block Jacobi case however the eigenvalues lie ten times further away from one then for $d = 2$, see figure A.11. The circulant preconditioner does not show this same behaviour, there is no dependence on the equilibrium depth.

3.3.6 Theory Eigenvalues Block Jacobi Case

For the block Jacobi case we can get an impression of the eigenvalues of the preconditioned matrix by assuming $A(x_i)$ are constant and equal to A. Then both J and W can be written in Kronecker product form:

$$J = (-I_k \otimes \Delta t A) + (C_k \otimes B) \tag{3.20}$$

$$W = (-I_k \otimes \Delta t A) + (I_k \otimes B). \tag{3.21}$$

These are both pre-multiplied by $I_k \otimes A^{-1}$, this gives

$$\hat{J} = \Delta t I + (C_k \otimes A^{-1}B) \tag{3.22}$$

$$\hat{W} = \Delta t I + (I_k \otimes A^{-1}B). \tag{3.23}$$

Let λ_{AB} be an eigenvalue of $A^{-1}B$ and λ_C an eigenvalue of C_k then the eigenvalue problem transforms into

$$-\Delta t + \lambda_C \lambda_{AB} - \lambda(-\Delta t + \lambda_{AB}) = 0. \tag{3.24}$$

From equation 2.17 we have $\lambda_C = 1 - e^{ij\Delta t}$, and we can now write

$$\lambda(-\Delta t + \lambda_{AB}) = -\Delta t + \lambda_{AB} - e^{\frac{ij2\pi}{k}}\lambda_{AB}, \tag{3.25}$$

thus

$$\lambda = 1 - \frac{e^{\frac{ij2\pi}{k}}\lambda_{AB}}{-\Delta t + \lambda_{AB}}. \tag{3.26}$$

This defines a family of circles around one with radius $|\frac{\lambda_{AB}}{-\Delta t + \lambda_{AB}}|$. It explains the circles discussed in the previous section.

3.4 Discussion of the Results

From the results above we can estimate how well our new preconditioning technique works. We can see that in the cases we expected the circulant preconditioner to perform better it really does, however the difference is not very big in the first simulations with this toy model. We saw that there are less iterations needed in the GMRES process, and on top of that it is clear that the runtimes become smaller for the circulant preconditioner if the FFT is applied. The eigenspectra we studied support this conclusion, since there is little dependence on Δt for the circulant preconditioner, in contrary to block Jacobi, which shows a bigger dependence on the

value of Δt in the eigenspectra. In the situation with a spin up time there is a much more clear difference between the block Jacobi and the Circulant/FFT approach. We see that the used CPU time as well as the number of GMRES iterations become much larger in this case then in the situation without spin up and the Circulant preconditioner is much faster and needs about the same number of iterations as in the simple situation without spin up.

This gives an indication that for the much more complicated ocean model the circulant preconditioner may have advantages in solving for periodic orbits and therewith it can help us do more efficient numerical continuation and bifurcation analysis of the ocean equations.

Chapter 4

Ocean Model

Ocean modelling and simulation is of great interest these days. It gives results which are of interest for example in climate research. There are a lot of people involved in this research. There are a lot of choices to be made when constructing an ocean model. For climate research we wish to be able to look at perturbations of the ocean dynamics. Due to (global) warming, we can for example expect the North Atlantic Current to change, which would have a big effect on our climate [19].

Why we need a model, is because of the complexity of the global ocean flow. The complexity comes from the domain which has a far from trivial shape, since we have the continents and a bottom with steep slopes. Another factor is the Coriolis force. Furthermore there are a number of quantities which drive the flow:

- Wind
- Temperature
- Salinity
- Inflow from rivers and melting ice

In the past various models have been created. The THCM model has been developed at the university of Utrecht, by the Institute of Marine and Atmospheric research. This model computes six quantities by numerically computing the solution to the nonlinear system of partial differential equations that describe the dynamics of the system and therewith the relations between the six quantities which are: flow velocity in three directions, pressure, temperature and salinity.

Although modern computers have a lot of computational capacity it still takes a lot of time and memory to compute the solutions to differential equations involved in the ocean model. Furthermore we are especially interested in the effect of changing parameters (numerical continuation) which will give us an idea of the change of the flow caused by a change in the driving forces.

Since a seasonal forcing is applied at the ocean surface in our model (wind, temperature and a virtual salt flux) the forcing is based on interpolated monthly mean data, the resulting flow is then periodic in time. This means we can try to solve it by applying the method described in chapter two of this thesis. Before, it has been tried to apply a Block-Jacobian preconditioner, this only converges if we do not take into account the temperature and salinity forcing, which is a huge drawback. If we manage to solve the model with all the forcings we can try numerical continuation and look for branches and bifurcations. In our case we are looking for steady states

in the 4-D system and the solutions we get are solutions for a certain space and time domain, in other words we are looking for periodic orbits in the ocean model.

4.1 Ocean Model Characteristics and Equations

4.1.1 Equations

The equations solved in the THCM ocean model are a set of coupled nonlinear partial differential equations. They describe the dynamics in six variables, which are zonal (u), meridional (v) and vertical velocity (w), furthermore we have the dynamic pressure (p) and the temperature (T) and salinity (S). Because of the spherical nature of the earth's shape we will describe the equations in spherical coordinates. These can be found in [9] and are given by

$$\frac{Du}{dt} - uv\tan(\Theta) - 2\Omega v\sin(\Theta) + \frac{1}{\rho_0 r_0\cos\Theta}\frac{\partial p}{\partial\phi} = A_V\frac{\partial^2 u}{\partial z^2} + A_H L_u(u,v) + \frac{\tau_0}{\rho_0 H_m}\tau^{\phi}G(z) \quad (4.1)$$

$$\frac{Dv}{dt} - u^2\tan(\Theta) - 2\Omega u\sin(\Theta) + \frac{1}{\rho_0 r_0}\frac{\partial p}{\partial\Theta} = A_V\frac{\partial^2 v}{\partial z^2} + A_H L_v(u,v) + \frac{\tau_0}{\rho_0 H_m}\tau^{\Theta}G(z) \quad (4.2)$$

$$\frac{\partial p}{\partial z} = \rho_0 g(\alpha_T T - \alpha_S S) \quad (4.3)$$

$$\frac{\partial w}{\partial z} + \frac{1}{r_0 cos\Theta}\left(\frac{\partial u}{\partial\phi} + \frac{v(\cos\Theta)}{\partial\Theta}\right) = 0 \quad (4.4)$$

$$\frac{DT}{dt} - R(\rho)T = \frac{T_S - T}{\tau_T}G(z) \quad (4.5)$$

$$\frac{DS}{dt} - R(\rho)S = \frac{S_S - S}{\tau_S}G(z) \quad (4.6)$$

$$G(z) = H\left(\frac{z}{H_m} + 1\right) \quad (4.7)$$

$$\frac{D}{dt} = \frac{\partial}{\partial t} + \frac{u}{r_0\cos\Theta}\frac{\partial}{\partial\phi} + \frac{v}{r_0}\frac{\partial}{\partial\Theta} + w\frac{\partial}{\partial z} \quad (4.8)$$

$$L_u(u,v) = \nabla_H^2 u + \frac{u}{r_0^2\cos^2\Theta} - \frac{2\sin\Theta}{r_0^2\cos^2\Theta}\frac{\partial v}{\partial\phi} \quad (4.9)$$

$$L_v(u,v) = \nabla_H^2 v + \frac{v}{r_0^2\cos^2\Theta} + \frac{2\sin\Theta}{r_0^2\cos^2\Theta}\frac{\partial u}{\partial\phi} \quad (4.10)$$

$$\nabla_H^2 = \frac{1}{r_0^2\cos\Theta}\left[\frac{\partial}{\partial\phi}\left(\frac{1}{\cos\Theta}\frac{\partial}{\partial\phi}\right) + \frac{\partial}{\partial\Theta}\left(\cos\Theta\frac{\partial}{\partial\Theta}\right)\right] \quad (4.11)$$

$G(z) = H(z/H_m + 1)$ with H a continuous approximation to the Heaviside function. We also have the constant heat capacity C_p, τ_T and τ_S are the surface adjustment time scales of heat and salt respectively. Furthermore A_H and A_V are the horizontal and vertical momentum viscosity. and will be taken constant. $R(\rho)$ is the operator that takes care of the mixing of heat and salt, some more information is in the convective adjustment section. In the table below one finds the standard values of the parameters used.

Table 4.1: Standard Values of Parameters in the Ocean Model

$2\Omega = 1.4 \cdot 10^{-4}[s^{-1}]$	$r_0 = 6.4 \cdot 10^{6}[m]$
$\tau_T = 7.5 \cdot 10^{1}[days]$	$\tau_S = 7.5 \cdot 10^{1}[days]$
$\alpha_T = 1.0 \cdot 10^{-}4[K^{-1}]$	$\alpha_S = 7.6 \cdot 10^{-}4[-]$
$C_p = 4.2 \cdot 10^{3}[J(kgK)^{-1}]$	$1.0 \cdot 10^{3}[kgm^{-3}]$
$A_V = 1.0 \cdot 10^{-3}[m^2 s^{-1}]$	$A_H = 1.0 \cdot 10^{4}[m^2 s^{-1}]$

4.1.2 Boundary Conditions

Slip conditions are assumed on the bottom boundary, this means the normal velocity is zero at the bottom. At the other boundaries we have a no-slip condition, except for the ocean surface, in other words the velocity is zero relative to the lateral boundaries. At all boundaries the the heat flux is zero. At the surface the forcing is respresented as a body force over the first layer of the ocean, from which we have slip and no flux conditions. We can write this in formula's as

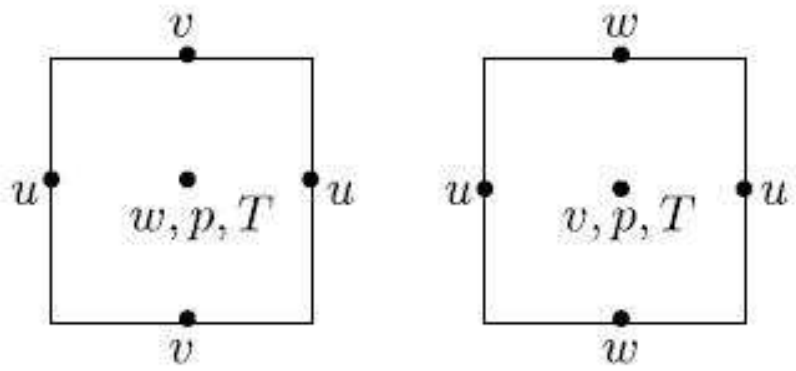
$$z = -D, 0 \ : \ \frac{\partial u}{\partial z} = \frac{\partial v}{\partial z} = w = \frac{\partial T}{\partial z} = \frac{\partial S}{\partial z} = 0 \tag{4.12}$$

$$\phi = \phi_w, \phi_e \ : \ u = v = w = \frac{\partial T}{\partial \phi} = \frac{\partial S}{\partial \phi} = 0 \tag{4.13}$$

$$\Theta = \Theta_s, \Theta_n \ : \ u = v = w = \frac{\partial T}{\partial \Theta} = \frac{\partial S}{\partial \Theta} = 0 \tag{4.14}$$

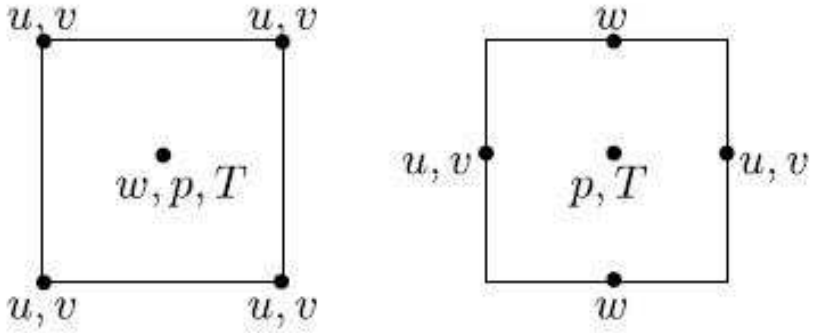
4.1.3 Discretisation

The equations in the ocean model are discretised in space by a second order accurate control volume discretisation method. In order to increase the number of grid points close to the surface the grid is stretched vertically, because most of the forcing is applied on the top layer (wind evaporation). In order to avoid troubles with horizontal viscosity, like wiggles, as described in [1] a B-grid is used in horizontal direction and a C-grid in vertical direction.



(a) Cgrid

Figure 4.1: Positioning of variables in the horizontal C-grid, topview (left) and vertical cross section (right).



(a) Bgrid

Figure 4.2: Positioning of variables in the horizontal B-grid, topview (left) and vertical cross section (right).

Because the equations only consist of derivatives of the pressure and the usage of the B-grid we have two degrees of freedom in the solution for the pressure, and therefore the pressure grid is a checkerboard. This is noticeable in the equations in the fact that the Jacobian has two singular vectors, or the nullspace has dimension two. As described in the section about complex

valued systems we solve a system twice the size and in both the real and imaginary system the same matrices are used and therefore we will find four degrees of freedom in the complex system, however the corresponding vectors that span the nullspace are the same for both the real and the imaginary part. In order to remove the checkerboard modes one has to subtract the nullspace from both the real and imaginary part.

The spatially discretised model equations can be written in the form

$$M\frac{dx}{dt} = F(x), \tag{4.15}$$

where x is a vector containing the six unknowns at every grid point. This means we already have a large system, if we for example take 8 grid cells in any direction we have $8 \times 8 \times 8 \times 6 = 3072$ unknowns, and still we took a small number of grid cells. Because of the correspondence with 2.1, we can apply the method described in chapter 2. If we do so the system to solve has a factor *number of time steps in a periodic orbit* more unknowns. This makes it very interesting to make use of parallel computing, especially because the proposed FFT's will give a complex system and thus we have twice the unknowns (real and imaginary part).

4.1.4 Convective Adjustment

In the equations convection is not taken into account. In case of unstable stratification, for example 'heavier' water is in a higher layer then water with less weight, we do want convection between those layers to take place, in order to avoid unphysical solutions. Therefore convective adjustment is brought in the model. It takes care of the vertical mixing of heat and salt if the flow becomes unstably stratified. This is done by increasing the vertical mixing coefficient of heat and salt in case of an unstable stratification.

4.1.5 Numerical Continuation

In general numerical continuation is performed on a set of algebraic equations of the form

$$F(x, \mu) = 0 \tag{4.16}$$

Where we want to investigate the influence of the parameter μ on the dynamics. In our case this is the factor for the forcing seasonal forcing in the THCM ocean model. The forcing is a combination of the forcings mentioned at the beginning of this chapter. Usually the parameter μ starts at zero, for which a solution is computed, and then μ is increased with a certain step size and with the previous solution as a starting guess a solution for the new value of the parameter is computed. This way one can search for a branch of steady solutions and find turning points or bifurcations on the branch. It is this way possible to tune certain parameters to their actual value and compute the effect of disturbances. Also it is possible to find out if there are multiple solutions for certain parameter values. In principle it helps us to find qualitative effects of the value of a parameter to the dynamics. In case of time stepping however it takes a lot of time to find these effects because of spin up time. This can be done faster using the four dimensional method with a good preconditioner.

Numerical continuation can be done in several ways and the method we used in the ocean model is natural continuation. This is a basic form of continuation where we use the solution for the previous value of the continuation parameter as an initial guess for the next one. In some cases a more sophisticated continuation method is favourable because the natural method is not able to handle turning points. For a more thorough description of numerical continuation methods and applications see [21] and [12]. The continuation is done by Trilinos package LOCA.

Next to the continuation method we have to choose a prediction method, this is the secant predictor, however since it needs two already computed steps we use a constant predictor for the first step. The corrector is Newton's method. Another factor in the continuation process is the step length control. It is sometimes usefull to adapt the step size of the continuation parameter during the process. If for example too large steps are taken the convergence of the Newton process will not converge. In other cases the proces may speed up by increasing the step size.

4.2 Implementation of the Preconditioner

The purpose of the Trilinos project is to facilitate the design, development, integration and ongoing support of mathematical software libraries. The main issue of the project is to develop parallel solvers for large scale complex multiphysics engineering and scientific applications. The available software is divided in packages. For our solver the packages Epetra, EpetraExt, Amesos, IfPack, NOX, LOCA and Komplex are of interest. Epetra and EpetraExt are the packages for the linear algebra items, such as vectors, matrices and linear systems. Amesos is the Direct Sparse Solver Package in Trilinos. We use this to solve the complex systems in the preconditioner. IfPack is a library with object-oriented algebraic preconditioners for the solution of preconditioned iterative solvers. NOX and LOCA are a combined package used for solving and analysing large nonlinear systems of equations. Komplex is a package that solves complex linear systems via equivalent real formulations.

The FFT Circulant preconditioner is build in the Trilinos THCM ocean model as class BlockCirculantSolver and is a preconditioner (Epetra) operator that can be used as an interface in NOX (NOX::Epetra::Interface::Preconditioner). For the FFT's the package FFTW (Fastest Fourier Transform in the West) is used. This is not a Trilinos package, we need it since there is no FFT available in Trilinos. Although our solver works parallel, the Fourier transform is not done parallel. Compared to the complex linear systems, the FFT's only take little time so that this step can be done sequentially.

4.3 Results of the Ocean Model

4.3.1 Settings

We used a tolerance of 10^{-6} for both the Krylov method (GMRES) and the Newton process. The simulations were executed on Huygens, the fastest super computer in the Netherlands. Next to the new developed preconditioner, also simulations were carried out using existing (XYZT-)preconditioners, these are described in section 2.4. In the simulations we used a $32 \times 32 \times 8$ grid with a corresponding land mask: North Atlantic, 2 degree resolution (32×32) continents and topography. The simulations are done on a domain bounded by

- Western domain bound 286 degrees
- Eastern domain bound 350 degrees
- Southern domain bound 10 degrees
- Northern domain bound 74 degrees

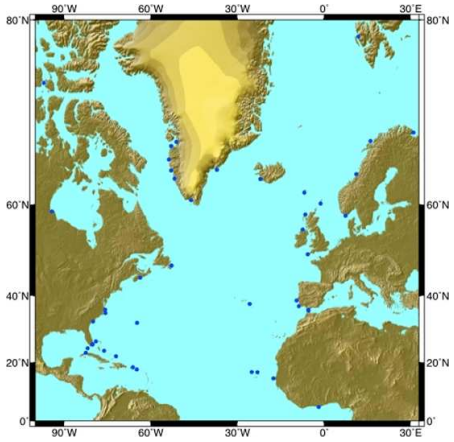


Figure 4.3: Domain of the Simulations

4.3.2 Forcing

At first we tried continuation of a periodic orbit with next to wind both temperature and salinity forcing turned on. This gave troubles in the Newton process after a few continuation steps. Also if we computed a starting solution in time independent setting this was not working. The first continuation steps are fine, but at some point the Newton process stagnates (no convergence in ten iterations), after which the continuation step size is halved. This is repeated several times until the step size becomes too small and the continuation stops. The reason for the stagnation turned out to be the convective adjustment, that leads to inconsistent systems in different time steps. Therefore we went to the case without salinity forcing and turned off the convective adjustment. The convective adjustment troubles are however expected to be solved in the near future and are not a problem for the block circulant solver only, therefore this is no reason to stop using the new method. First a starting solution is computed with a time independent continuation where the combined forcing of wind and temperature is slowly increased from

zero to one and seasonal forcing is turned off. With the starting solution the time dependent continuation process is started with seasonal forcing from zero to one.

4.3.3 Block Circulant preconditioner versus existing Trilinos preconditioners

Below four test cases are compared. The sequential solver is not parallel and therefore we run it on one process. The parareal is performed on 3 processes (4 time steps per process) just as parallel. And our own solver block circulant has a processor for every time step, which makes 12 processes in total. All these simulations are performed on one node. The Newton process needed at most 3 iterations in order to converge. We also ran the simulation with the block diagonal (Jacobi) preconditioner. This gives very slow convergence in the Krylov method. After 500 iterations we are still left with a residue of order 10^{-4} and it took far more than 300 iterations to improve the residue by a factor 10. It also took about 20 minutes to get to this result which means it takes a lot more time in comparison with our circulant preconditioner.

Table 4.2: Results of the continuation process, Krylov (GMRES) iterations

Average	Block Circulant	Sequential	Parallel	Parareal
Number of GMRES iterations	108	126	360	360
Solution time (sec.)	646	2058	2422	1981
Time to compute Preconditioner (sec.)	28	54	23	20

The results in table 4.2 show quite clearly the advantages of the block circulant preconditioner. If we look at the number of GMRES iterations we can see that for sequential and block Circulant are not very different both around 130 (see figure 4.5), however time it takes to solve the system is much longer in the sequential case, this is due to the nonparallel computation in this solver. If we look at the Parareal solver we see about the same solve times as with sequential, around 30 minutes against only 10 minutes for the block circulant solver. Parareal thereby also uses a much larger number of iterations just as parallel, which is probably due to the dropping of the B-blocks in the preconditioner, because of that the periodic behaviour is suppressed and the solver needs more iterations and therewith more time to find the solution. Parallel and parareal show the same behaviour in number of iterations, which indicates that the extra work in parareal is of no importance in this setting. The calculation time for parallel is much longer, this is due to the choice to run it in shared mode, there can have been more users on this node which slows down the proces.

We can also see that at the start of the continuation process our own block circulant solver is very fast. After some time it goes to a somewhat steady case where the solves take about 10 to 15 minutes. The parareal solver could probably do a bit better for a better tuned case if the number of time domains is decreased or increased. We assume that increasing the number of time steps will not help since this gives the Parareal solver more characteristics of the block Jacobi which already had bad convergence, the same goes for parallel. It seems unlikely that they will be able to compete with our solver.

Because it does not use parallel computing the sequential preconditioner takes a lot more time to compute. Parareal is quicker in computing the preconditioner, but since the difference is far from the order of the difference in solution time this is not worth mentioning.

The THCM ocean model allows next to time parallelisation also for parallelisation in space, with which it should be possible to speed up the continuation even more, this is not of much use in the relative small problems we considered in this research. We did run the same setting for 24

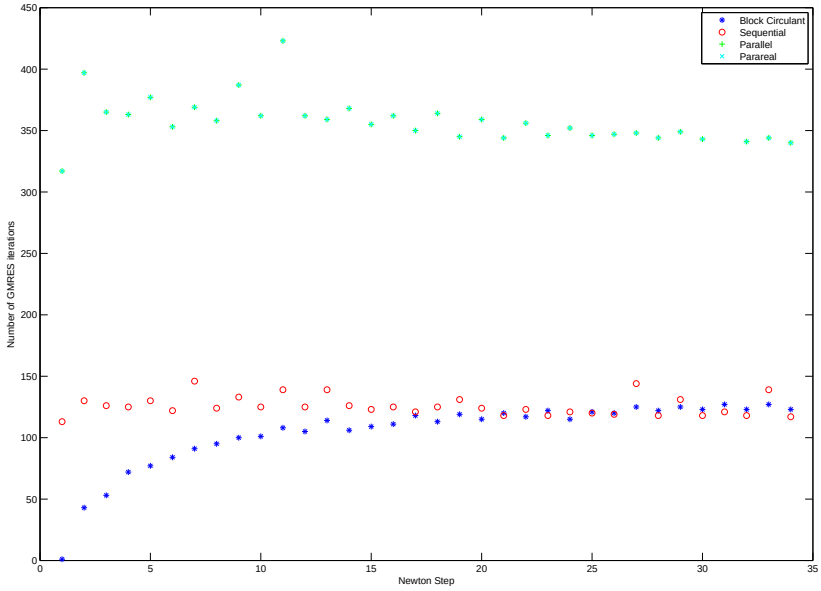


Figure 4.4: Comparison of the number of iterations in the continuation process for the different methods

time steps for which the results are in table 4.3. We can see that in average about 16 per cent more iterations are needed. The time per continuation step increases by about the some factor, which means that the time per iteration is about constant for a different number of time steps.

Table 4.3: Results of the continuation process

time steps	12	24
Number of GMRES iterations	108	163
Solution time (sec.)	646	1065
Time to compute Preconditioner (sec.)	28	29

4.4 Discussion of the Results

The results from the continuations in the ocean model are good for our method. We have seen that it needs much less iterations then other parallel preconditioners and also less time is needed for all cases. Unfortunately we could not use both temperature and salinity forcing and also convective adjustment which takes care of the mixing is turned off. We worked without salinity forcing and for this case the block circulant method works very well, also parallel.

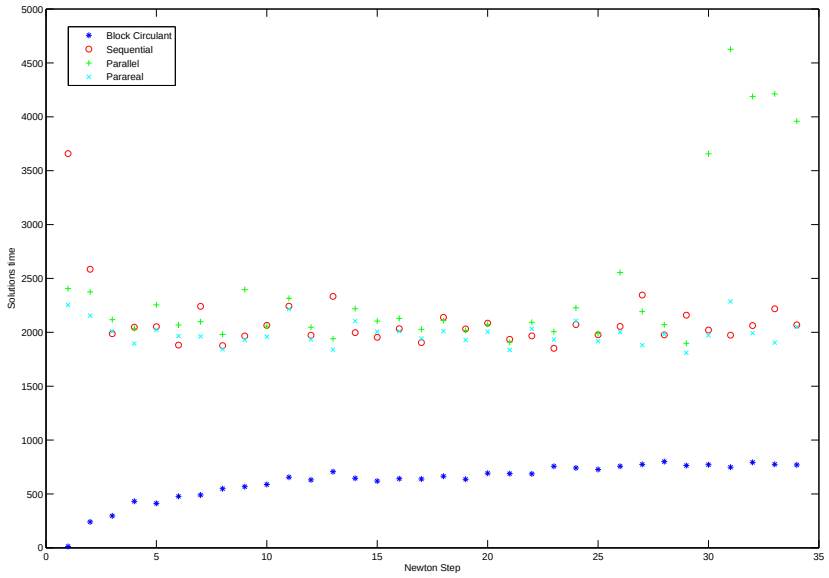
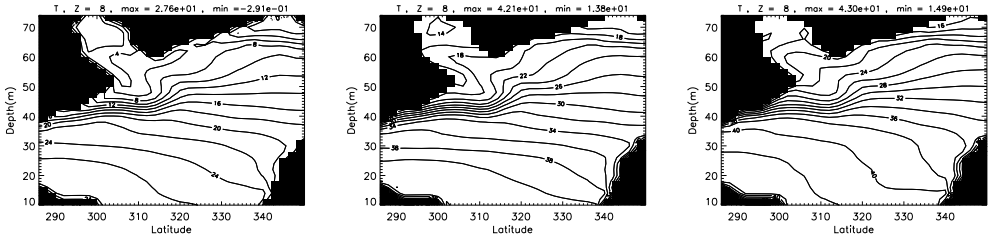


Figure 4.5: Comparison of the solution times in the continuation process for the different methods

4.5 Solutions of the Ocean Model

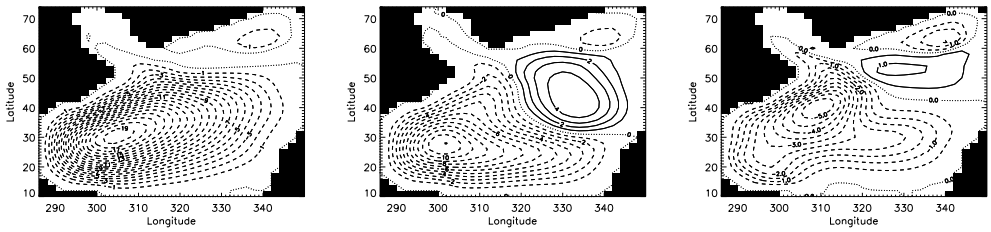
The solutions of the model are visible in the following figures. We looked at the starting solutions and the solutions for the months January and July in order to make sure that the solutions are indeed different then without seasonal forcing. We can see in figure 4.6 that the temperature forcing indeed has influence in the solutions. In both the January and the July solution the temperature is much different from the starting solution. The January and July temperature distributions show higher temperatures in summer then in winter months, with a difference of about two degrees. Note that the solutions can not be expected to be realistic because of the lack of salinity and convection in the test problem.



(a) Starting Solution (b) January (c) July

Figure 4.6: Temperature distribution top layer

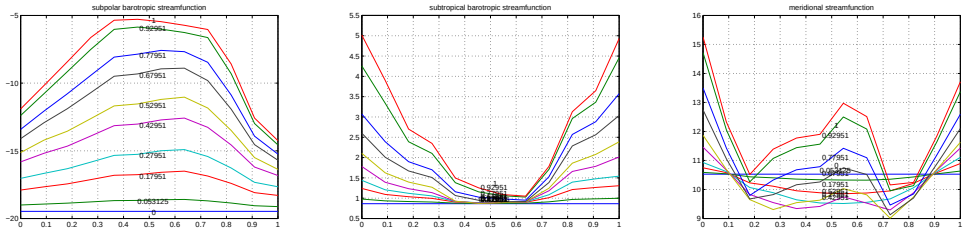
In 4.7 we see the patterns of the barotropic streamfunction. We see some clear difference in the upper right corner, where an extra circular pattern occurs in comparison with the starting



(a) Starting Solution (b) January (c) July

Figure 4.7: Barotropic streamfunction top layer

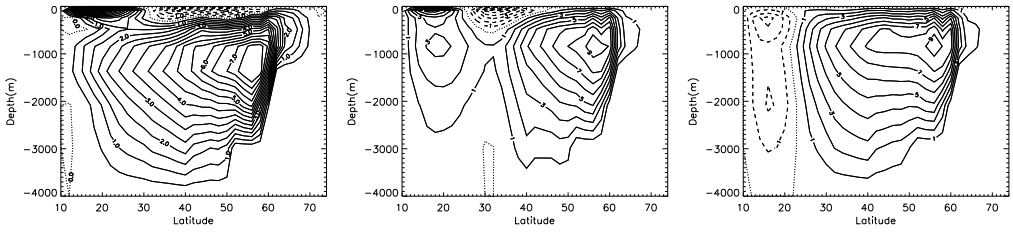
solution. It is visible in both the January and July solution, although these solution also show a difference again. The periodic behaviour of these solutions becomes clear in 4.8 where we see the subpolar and subtropical barotropic streamfunction during a year, both show periodic behaviour, indicating we are dealing with a periodic solution of the flow. The numbers in the figure are the magnitude of the continuation parameter.



(a) Subpolar Barotropic (b) Subtropical Barotropic (c) Meridional

Figure 4.8: Streamfunction

From figure 4.8 we can see that for increasing continuation parameter the behaviour becomes more and more periodic, indicating that the seasonal forcing does it's work. The meridional streamfunction also shows periodic behaviour. The solutions for a cross section can be seen in figure 4.9.



(a) Starting Solution (b) January (c) July

Figure 4.9: Meridional streamfunction cross section

Chapter 5

Discussion

The goal of this thesis was to implement and investigate the performance of the block circulant preconditioner especially on the ocean model. First we applied it to the shallow water equations. The simulations were in this case done with matlab. The results for the situation in which we took the scaling more or less equal to ocean dimensions. This gave not completely convincing results. For the block circulant preconditioner the number of iterations in GMRES did not depend on the time step, however for large time steps block Jacobi does better and for smaller time steps there is not much difference. Most likely these competitive results are caused by the absence of a spin up time in this setting. Therefore we also did experiments in which a spin up time is present. This way we found results which are very much in favour of the block circulant method. The number of iterations as well as the needed CPU time were much smaller then for block Jacobi. Since for the Ocean model there is a spin up time, we concluded that this might work very well in the THCM ocean model too.

The implementation in the ocean model took a lot of time with some unexpected obstacles, but the results are quite good. We found that in solution time the existing methods which are not able to make use of parallel computing, or at least not as much as the new method, needed about two to four times as long. The block Jacobi which does allow to compute every time step in parallel did not converge at all in a reasonable time if temperature forcing is turned on. Therefore it seams reasonable to conclude that the block circulant preconditioner can be applied in more cases in which periodic orbits are common. However we did not manage to make the ocean model work with next to temperature also salinity forcing turned on due to problems with vertical mixing. Also convective adjustment which takes care of this mixing has been turned off in our simulations. Because these things also didn't work with existing methods we can conclude that the block circulant preconditioner gives us better results then before. The time needed for a continuation process is still long, since we only used $32 \times 32 \times 8$ grid points for the North Atlantic region it will be a lot longer for larger regions, nevertheless the new method works much faster then existing methods and the parallel possibilities make it interesting to apply this method to even larger problems. The technique can be even more exploited if we go to complex calculations, because then we would not need systems of twice the dimensions as described in section 2.5, which has big advantages in storage.

Appendices

Appendix A

Eigenspectra

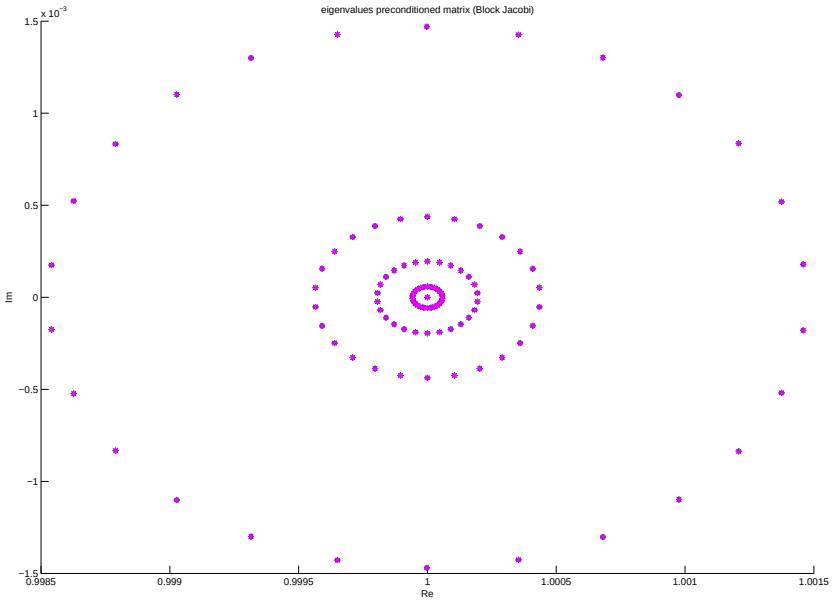


Figure A.1: Eigenspectra of block Jacobi preconditioned matrices for two succeeding Newton iterations

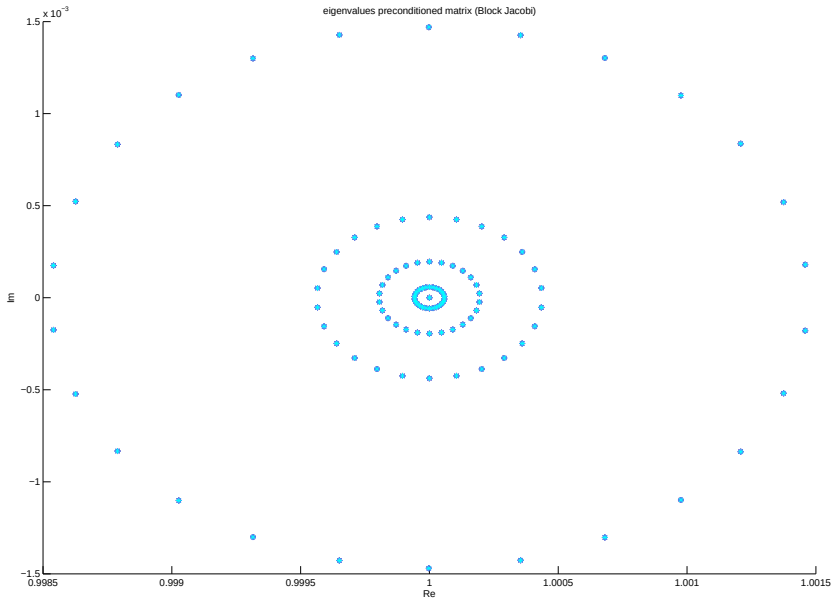


Figure A.2: Eigenspectra of block Jacobi preconditioned matrices for two succeeding Newton iterations

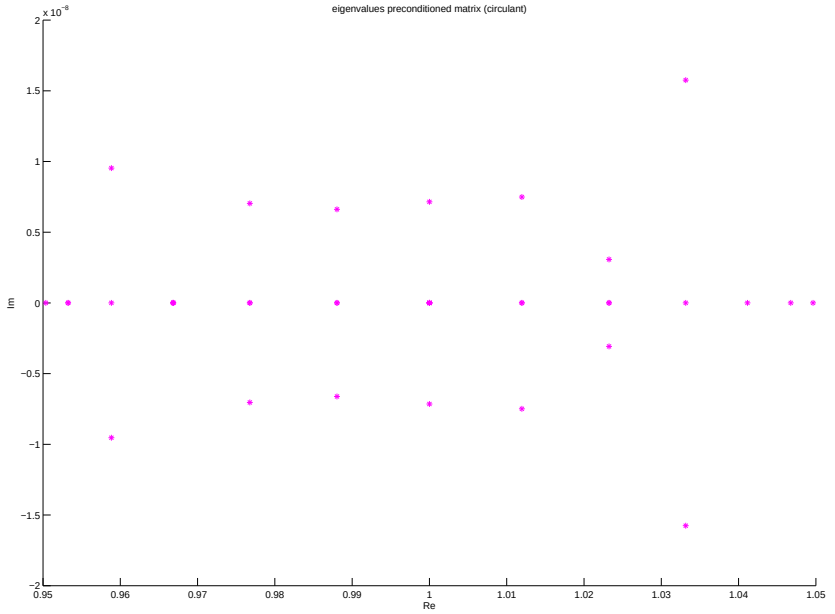


Figure A.3: Eigenspectra of circulant preconditioned matrices for two succeeding Newton iterations

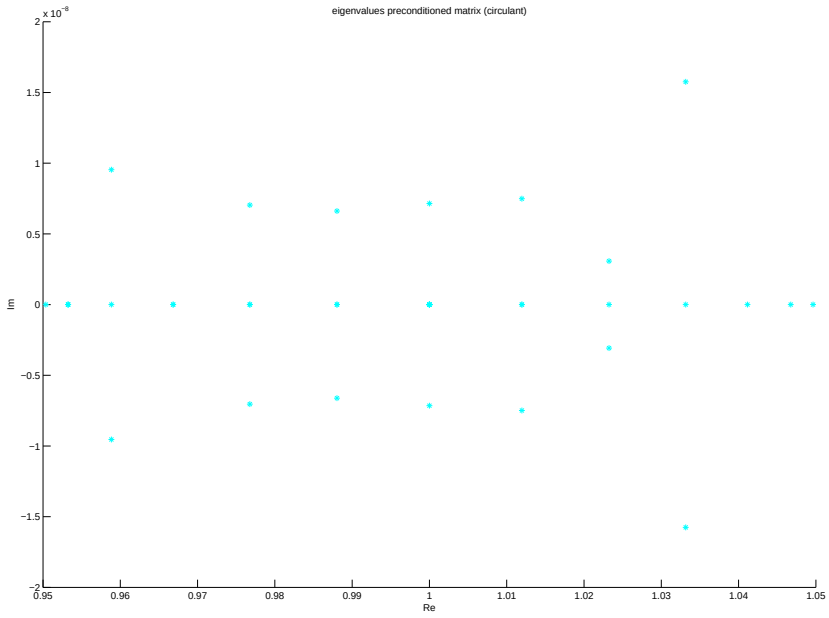
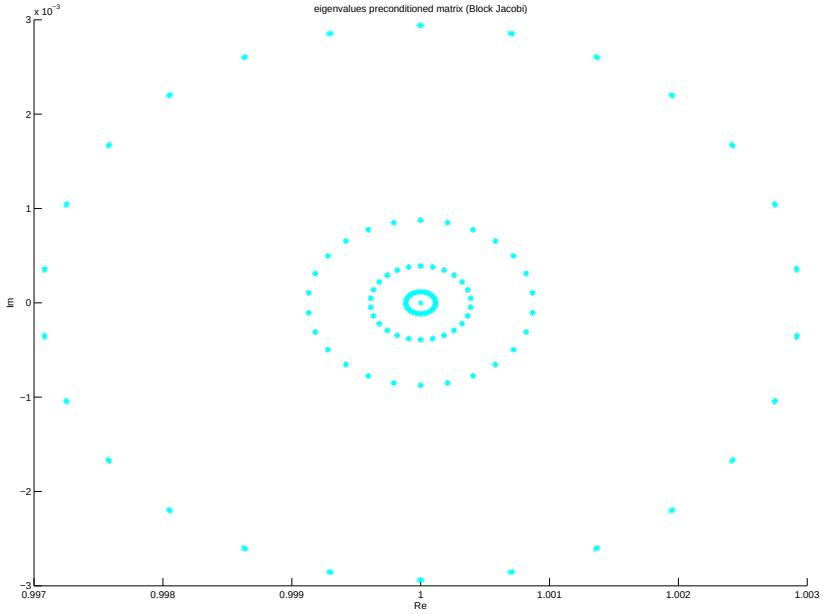


Figure A.4: Eigenspectra of circulant preconditioned matrices for two succeeding Newton iterations



(a) BJ3

Figure A.5: Eigenspectra for $\Delta t = 2$ of block Jacobi

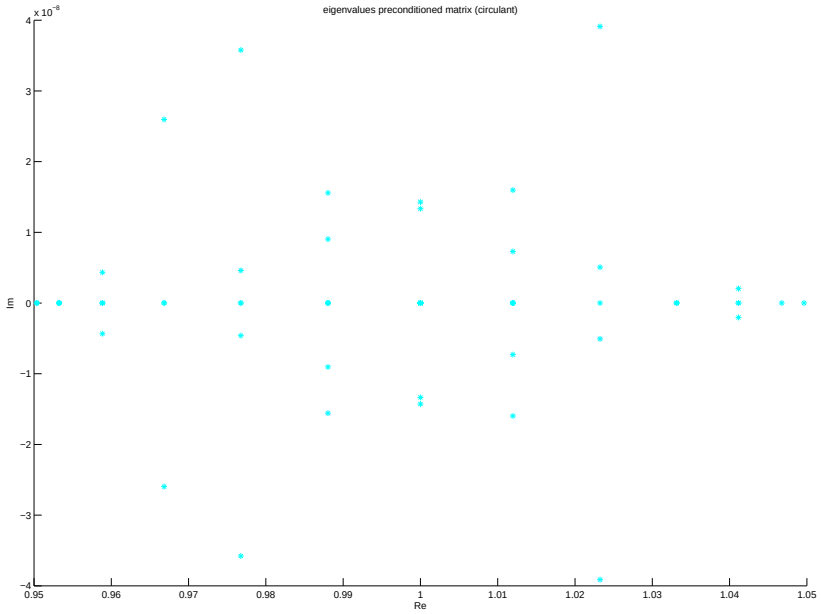


Figure A.6: Eigenspectra for $\Delta t = 2$ of circulant

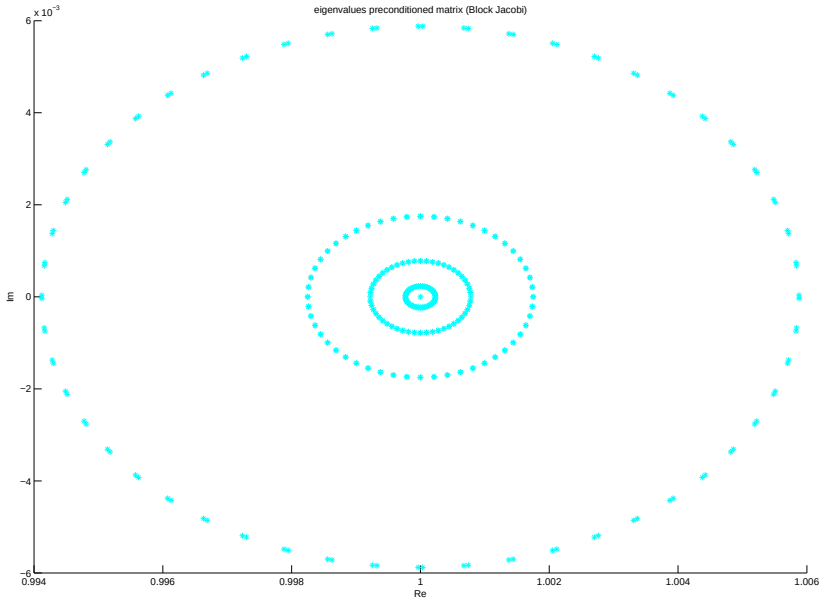


Figure A.7: Eigenspectra for $\Delta t = 1$ of block Jacobi

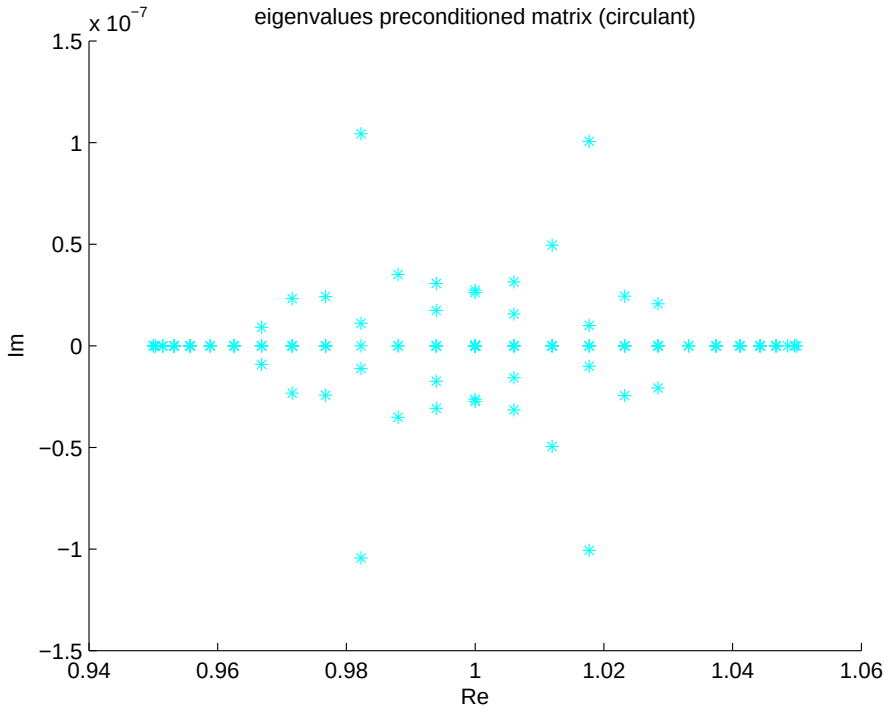


Figure A.8: Eigenspectra for $\Delta t = 1$ of circulant

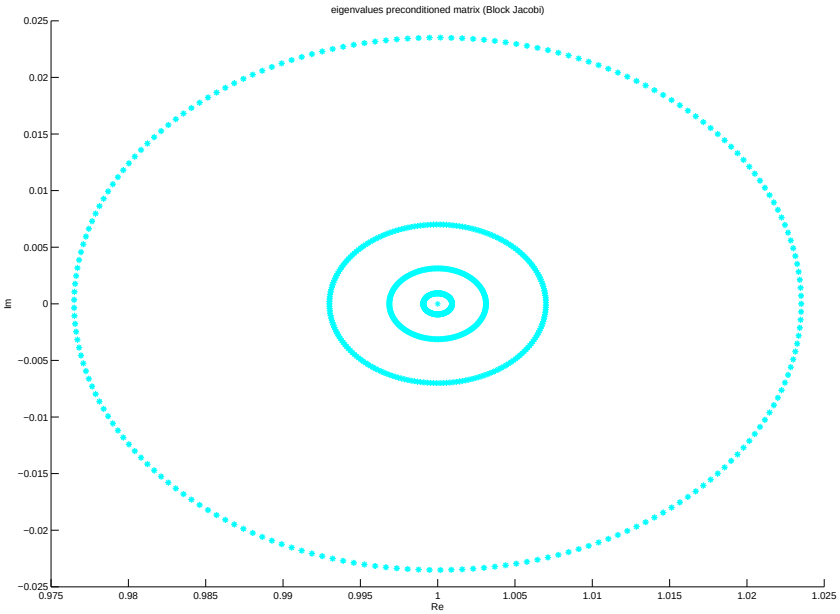


Figure A.9: Eigenspectra for $\Delta t = 0.25$ of block Jacobi

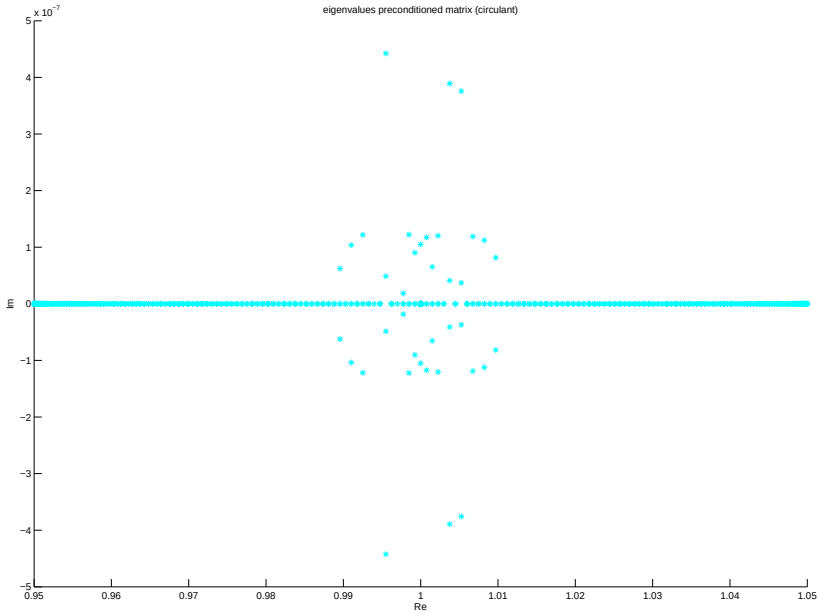


Figure A.10: Eigenspectra for $\Delta t = 0.25$ of circulant

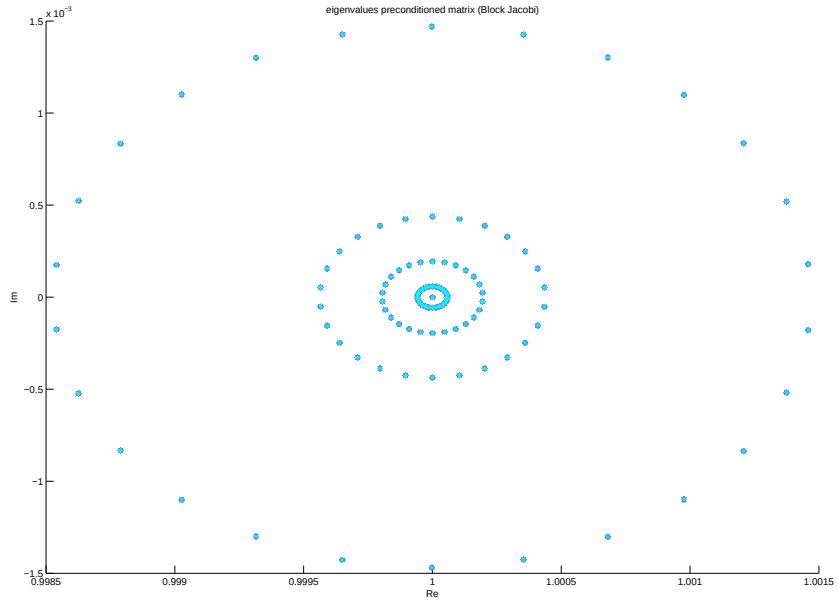


Figure A.11: Eigenspectra block Jacobi for $\Delta t = 1$ for $d = 2$

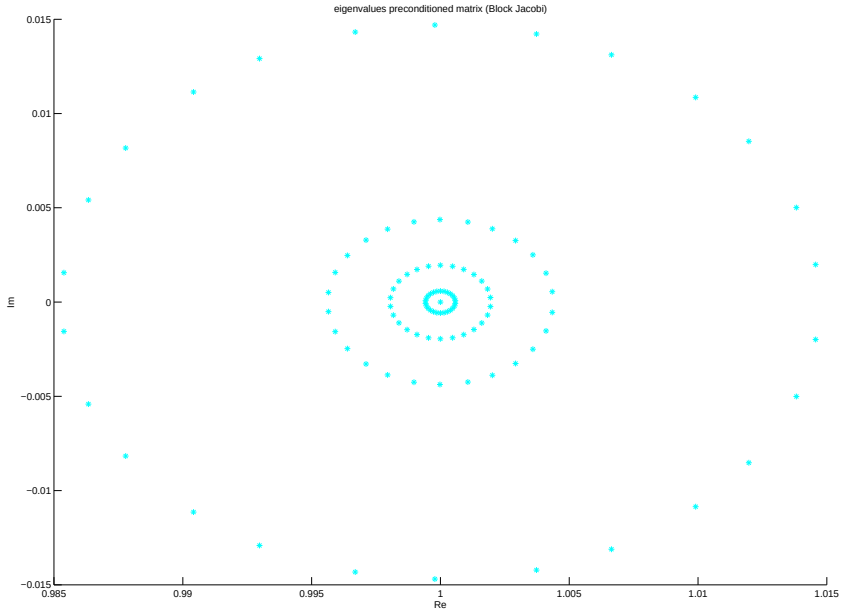


Figure A.12: Eigenspectra block Jacobi for $\Delta t = 1$ for $d = 0.02$

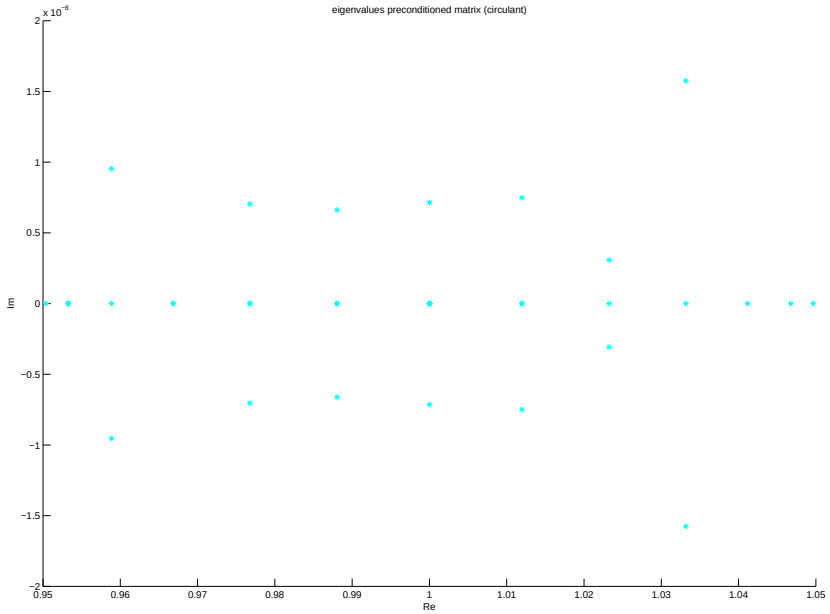


Figure A.13: Eigenspectra circulant for $\Delta t = 1$ for $d = 2$

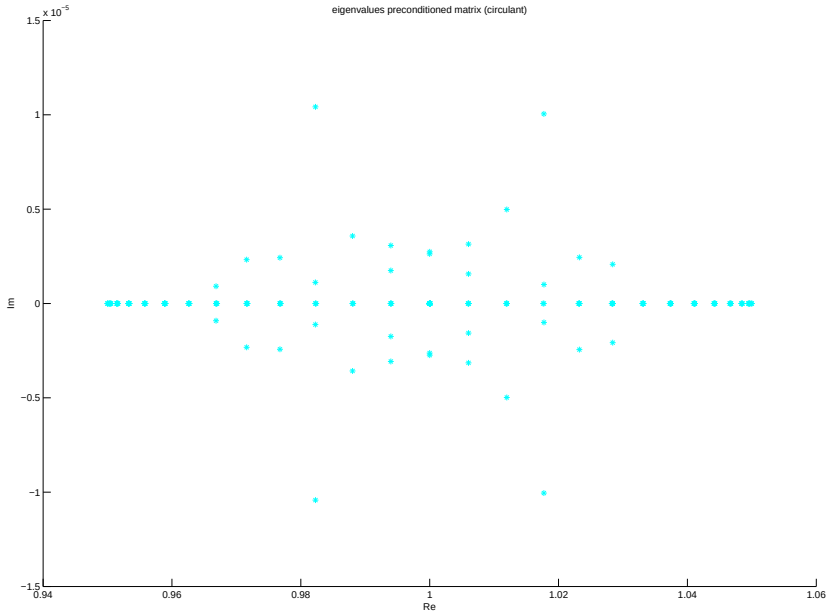


Figure A.14: Eigenspectra circulant for $\Delta t = 1$ for $d = 0.02$

Bibliography

[1] Fred W. Wubs Arie C. de Niet, Henk A. Dijkstra. The performance of implicit ocean models on b- and c-grids. *Journal of Computational Physics*, (211):210–228, December 2005.

[2] Arjen Terwisscha van Scheltinga Henk A. Dijkstra Arie de Niet, Fred Wubs. A tailored solver for bifurcation analysis of ocean-climate models. *Journal of Computational Physics*, (227):654–679, August 2007.

[3] Peter G. Baines. *Topographic effects in stratified flows.* Cambridge University Press, 1995.

[4] E. Oran. Brigham. *The Fast Fourier Transform.* Prentice Hall, Inc., Englewood Cliffs, New Jersey., 1974.

[5] H.W. Broer. Dynamische systemen en chaos.

[6] Raymond H. et al. Chan. Fft-based preconditioners for toeplitz-block least squares problems. *Siam J. Numer. Anal.*, 30(6):1740–1768, December 1993.

[7] Mingkui. Chen. On the solution of circulant linear systems. *Siam J. Numer. Anal.*, 24(3):668–683, June 1987.

[8] David Day and Michael A. Heroux. Solving complex-valued linear systems via equivalent real formulations. *Siam J. Sci. Comput.*, 23(2):480–498, 2001.

[9] Arie de Niet. *Solving Large Linear Systems in an Implicit Thermohaline Ocean Model.* Rijksuniversiteit Groningen, 2007.

[10] Michael Heroux et al. An overview of trilinos. August 2003.

[11] S.M. Griffies et al. Formulation of an ocean climate model. *Ocean Science*, (1):45–79, 2005.

[12] Kurt Georg Eugene L. Allgower. *Numerical Continuation Methods, An Introduction.* Springer-Verlag, 1990.

[13] H. Eves. *Elementary Matrix Theory.* Dover Publications, 1980.

[14] J.W. Greidanus. Continuation of a single layer flow over an obstacle. 2008.

[15] Fred Wubs Eugen F.F. Botta Henk A. Dijkstra, Hakan Oksuzoglu. A fully implicit ocean model of the three-dimensional thermohaline ocean circulation. *Journal of Computational Physics*, (173):685–715, August 2001.

[16] Michael A. Heroux and James M. Willenbring. Trilinos users guide. September 2007.

[17] Lee R. Nackman John J. Barton. *Scientific and Engineering C++.* Addison Wesley, 1994.

[18] Steven G. Johnson Matteo Frigo. Fftw documentation 3.2.2.

[19] Stefan Rahmstorf. Thermohaline ocean circulation. *Encyclopedia of Quaternary Sciences*, 2006.

[20] J. Douglas Faires Richard L. Burden. *Numerical Analysis 8th edition.* Thomson Brooks/Cole, 2005.

[21] Rudiger Seydel. *Practical Bifurcation and Stability Analysis, Third Edition.* Springer, 2010.

[22] F.W. Wubs. Computational methods of science.

[23] Mayday Y. The parareal in time algorithm. 2008.