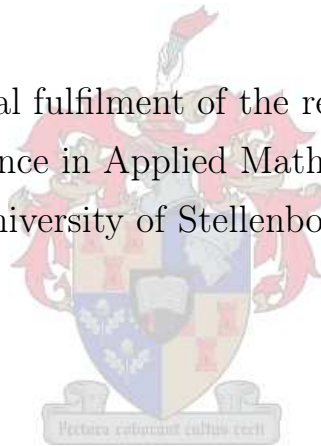


# Efficient Computation of Shifted Linear Systems of Equations with Application to PDEs

by  
Eyaya Birara Eneyew

Thesis presented in partial fulfilment of the requirements for the degree  
Master of Science in Applied Mathematics at the  
University of Stellenbosch



Supervisor: Prof. J.A.C. Weideman  
Faculty of Science  
Department of Mathematical Sciences

December 2011

# Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.



Signature: \_\_\_\_\_

Eyaya Birara Eneyew

Date: \_\_\_\_\_ September 27, 2011

Copyright © 2011 Stellenbosch University  
All rights reserved.

# Abstract

In several numerical approaches to PDEs shifted linear systems of the form

$$(zI - A)\mathbf{x} = \mathbf{b},$$

need to be solved for several values of the complex scalar  $z$ . Often, these linear systems are large and sparse. This thesis investigates efficient numerical methods for these systems that arise from a contour integral approximation to PDEs and compares these methods with direct solvers.

In the first part, we present three model PDEs and discuss numerical approaches to solve them. We use the first problem to demonstrate computations with a dense matrix, the second problem to demonstrate computations with a sparse symmetric matrix and the third problem for a sparse but nonsymmetric matrix. To solve the model PDEs numerically we apply two space discretization methods, namely the finite difference method and the Chebyshev collocation method. The contour integral method mentioned above is used to integrate with respect to the time variable.

In the second part, we study a Hessenberg reduction method for solving shifted linear systems with a dense matrix and present numerical comparison of it with the built-in direct linear system solver in SciPy. Since both are direct methods, in the absence of roundoff errors, they give the same result. However, we find that the Hessenberg reduction method is more efficient in CPU-time than the direct solver. As application we solve a one-dimensional version of the heat equation.

In the third part, we present efficient techniques for solving shifted systems with a sparse matrix by Krylov subspace methods. Because of their shift-invariance property, the Krylov methods allow one to obtain approximate solutions for all values of the parameter, by gen-

erating a single approximation space. Krylov methods applied to the linear systems are generally slowly convergent and hence preconditioning is necessary to improve the convergence. The use of shift-invert preconditioning is discussed and numerical comparisons with a direct sparse solver are presented. As an application we solve a two-dimensional version of the heat equation with and without a convection term. Our numerical experiments show that the preconditioned Krylov methods are efficient in both computational time and memory space as compared to the direct sparse solver.

# Opsomming

In verskeie numeriese metodes vir PDVs moet geskuifde lineêre stelsels van die vorm

$$(zI - A)\mathbf{x} = \mathbf{b}, \quad (1)$$

opgelos word vir verskeie waardes van die komplekse skalaar  $z$ . Hierdie stelsels is dikwels groot en yl. Hierdie tesis ondersoek numeriese metodes vir sulke stelsels wat voorkom in kontoerintegraalbenaderings vir PDVs en vergelyk hierdie metodes met direkte metodes vir oplossing.

In die eerste gedeelte beskou ons drie model PDVs en bespreek numeriese benaderings om hulle op te los. Die eerste probleem word gebruik om berekenings met 'n vol matriks te demonstreer, die tweede probleem word gebruik om berekenings met yl, simmetriese matrikse te demonstreer en die derde probleem vir yl, onsimmetriese matrikse. Om die model PDVs numeries op te los beskou ons twee ruimte-diskretisasie metodes, naamlik die eindige-verskilmetode en die Chebyshev kollokasie-metode. Die kontoerintegraalmetode waarna hierbo verwys is word gebruik om met betrekking tot die tydveranderlike te integreer.

In die tweede gedeelte bestudeer ons 'n Hessenberg ontbindingsmetode om geskuifde lineêre stelsels met 'n vol matriks op te los, en ons rapporteer numeriese vergelykings daarvan met die ingeboude direkte oplosser vir lineêre stelsels in SciPy. Aangesien beide metodes direk is lewer hulle dieselfde resultate in die afwesigheid van afrondingsfoute. Ons het egter bevind dat die Hessenberg ontbindingsmetode meer effektief is in terme van rekenaartyd in vergelyking met die direkte oplosser. As toepassing los ons 'n een-dimensionele weergawe van die hittevergelyking op.

In die derde gedeelte beskou ons effektiewe tegnieke om geskuifde stelsels met 'n yl matriks

op te los, met Krylov subruimte-metodes. As gevolg van hul skuifinvariansie eienskap, laat die Krylov metodes mens toe om benaderde oplossings te verkry vir alle waardes van die parameter, deur slegs een benaderingsruimte voort te bring. Krylov metodes toegepas op lineêre stelsels is in die algemeen stadig konvergerend, en gevolglik is prekondisionering nodig om die konvergensie te verbeter. Die gebruik van prekondisionering gebaseer op skuif-en-omkeer word bespreek en numeriese vergelykings met direkte oplossers word aangebied. As toepassing los ons 'n twee-dimensionele weergawe van die hittevergelyking op, met 'n konveksie term en daarsonder. Ons numeriese eksperimente dui aan dat die Krylov metodes met prekondisionering effektief is, beide in terme van berekeningstyd en rekenaargeheue, in vergelyking met die direkte metodes.

# Dedications

*To Jerry love*

# Acknowledgements

First and foremost, I would like to thank the Almighty God for his blessings and giving me the strength to finish this thesis.

I am grateful to my supervisor Prof. J.A.C. Weideman for his enormous support and guidance, thoughtful suggestions, practical advice, insightful direction, objective comments and encouragement at every stage of this thesis. Thank you for always understanding what I meant even when I was not able to phrase it. Without your knowledge and expertise this thesis would never have been completed. I would also like to express my appreciation to Dr. S.J. van der Walt for his continuous help and answering my questions on SciPy.

I would like to thank AIMS and the Department of Mathematical Sciences at the University of Stellenbosch for financial support.

My wholehearted thanks to Ethiopian folks Abey Sherif, Amare Abebe, Bewuketu Teshale, Daniel Gebeyehu and Dr. Mulugeta Admasu for their friendship, advice, prayers and encouragement.

In addition, I would like to thank my office-mates Albert Swart, Brenden Andries, Erhardt De Kock, Fine Wilms, Janto Dreijer, Marèt Coete and Pierre Joubert who were always there for support and encouragement and making the duration of my thesis enjoyable and very quiet.

Last but not the least, I would like to thank all my family members for their love, prayers and moral support. Finally, I would like to thank my love Eyerus Birhan for her understanding and patience during my stay in South Africa.



# Contents

<b>Declaration</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Opsomming</b>	<b>iv</b>
<b>Dedication</b>	<b>vi</b>
<b>List of Figures</b>	<b>xii</b>
<b>List of Algorithms</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Description . . . . .	1
1.2 Motivation . . . . .	3
1.3 Organization of the thesis . . . . .	4
<b>2 Benchmark Problems</b>	<b>6</b>
2.1 Introduction . . . . .	6
2.2 Heat Equations . . . . .	6
2.2.1 The 1D Heat Equation . . . . .	6

2.2.2	The 2D Heat Equation . . . . .	8
2.3	Space Discretization Methods . . . . .	9
2.3.1	Finite Differences . . . . .	10
2.3.2	Chebyshev Collocation Spectral Method . . . . .	15
2.4	Time Integration Methods . . . . .	18
2.4.1	Contour Integral Method . . . . .	18
2.4.2	The Trapezoidal Rule . . . . .	19
<b>3</b>	<b>Dense shifted linear systems</b>	<b>21</b>
3.1	Introduction . . . . .	21
3.2	The Hessenberg Reduction . . . . .	22
3.2.1	Householder Reflections . . . . .	22
3.2.2	The Hessenberg Reduction via Householder Reflections . . . . .	23
3.2.3	Operation Count for Hessenberg Reduction. . . . .	26
3.2.4	Explicit Computation of the Orthogonal Matrix . . . . .	27
3.3	Hessenberg Reduction Method . . . . .	30
3.4	Numerical Comparison of HRM and DLSS . . . . .	32
3.5	Summary and Conclusion . . . . .	34
<b>4</b>	<b>Sparse shifted linear systems</b>	<b>36</b>
4.1	Introduction . . . . .	36
4.2	Krylov Subspace Methods . . . . .	38
4.2.1	The Search Subspace . . . . .	39

4.2.2	The Subspace of Constraints . . . . .	43
4.2.3	Full Orthogonalization Method . . . . .	44
4.2.4	General Minimal Residual Method . . . . .	45
4.2.5	Lanczos Method . . . . .	47
4.2.6	Minimal Residual Method . . . . .	48
4.3	Shifted Linear Systems . . . . .	49
4.3.1	FOM for shifted linear systems . . . . .	50
4.3.2	GMRES for shifted linear systems . . . . .	51
4.3.3	Lanczos method for shifted linear systems . . . . .	51
4.3.4	MINRES for shifted linear systems . . . . .	51
4.4	Preconditioning . . . . .	52
4.5	Numerical Results . . . . .	57
4.5.1	Symmetric Problem . . . . .	57
4.5.2	Nonsymmetric Problem . . . . .	59
4.6	Summary and conclusion . . . . .	62
	<b>Appendix</b>	<b>64</b>
	<b>Bibliography</b>	<b>64</b>

# List of Figures

2.1	Analytical solution of the one-dimensional heat equation. . . . .	7
2.2	Analytical solution of the symmetric problem. . . . .	8
2.3	Analytical solution of the nonsymmetric problem . . . . .	9
2.4	A 2D grid with grid function values at its mesh points . . . . .	11
2.5	Schematic plot of parabolic contour . . . . .	18
3.1	CPU-time vs. order of matrix for the Hessenberg reduction . . . . .	29
3.2	Built-in <code>hessenberg</code> function: CPU-time vs. order of matrix . . . . .	30
3.3	Improved <code>hessenberg</code> function: CPU-time vs. order of matrix . . . . .	31
3.4	Numerical solution of the 1D heat equation model problem . . . . .	32
3.5	CPU-time vs. number of nodes of DLSS and HRM for different $N$ . . . . .	33
3.6	Total error in maximum norm vs. number of nodes of 1D heat equation . . . . .	34
3.7	Total error in the maximum norm vs. CPU-time(sec.) of 1D heat equation . . . . .	35
4.1	Spectrum of $z_2 I - A$ for the symmetric model problem . . . . .	53
4.2	Spectrum of the preconditioned matrix $A(z_2)$ for the symmetric model problem . . . . .	55
4.3	Spectrum of $A(z_k)$ , $k = 1, 2, 3, 4$ , for the symmetric problem . . . . .	55
4.4	Relative error vs. number of nodes on of LM, MINRES and DSS . . . . .	58

4.5	Relative error vs. number of nodes of PrecLM, PrecMINRES and DSS . . .	59
4.6	Relative error vs. CPU-time of the symmetric problem. . . . .	60
4.7	Relative error vs. number of nodes of FOM, GMRES and DSS . . . . .	60
4.8	Relative error vs. number of nodes of PrecFOM, PrecGMRES and DSS . . .	61
4.9	Relative error vs. CPU-time of the nonsymmetric problem. . . . .	62

# List of Algorithms

3.1	Computation of Householder vector . . . . .	24
3.2	Hessenberg reduction of a matrix . . . . .	26
3.3	Computation of $Q^T \mathbf{x}$ . . . . .	27
3.4	Computation of $Q\mathbf{x}$ . . . . .	27
3.5	Explicit computation of $Q$ . . . . .	29
4.1	General projection method . . . . .	39
4.2	Arnoldi process-Modified Gram-Schmidt . . . . .	41
4.3	Lanczos process . . . . .	42
4.4	Full Orthogonalization Method . . . . .	45
4.5	General Minimal Residual Method . . . . .	47
4.6	Lanczos Method . . . . .	48
4.7	Minimal Residual Method . . . . .	49

# Chapter 1

## Introduction

### 1.1 Problem Description

When linear parabolic partial differential equations (PDEs) are semidiscretized by the method-of-lines, systems of ordinary differential equations (ODEs) of the form

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u}(t), \quad \mathbf{u}(0) = \mathbf{u}_0, \quad (1.1)$$

are obtained. The matrix  $A$  is the result of approximating the space derivatives in the PDE. Two techniques for derivative approximation are finite difference formulas [12], which are local methods, and spectral methods [3, 22], which are global methods. Both are used in this thesis.

To approximate the solution of (1.1) one can use numerical ODE integrators, such as Runge-Kutta methods or multistep methods. However, in this thesis, instead of solving equation (1.1) using ODE integrators we use a contour integral method [26].

In principle, the solution of equations (1.1) is given in terms of the matrix exponential

$$\mathbf{u}(t) = \exp(At)\mathbf{u}_0. \quad (1.2)$$

Computing the matrix exponential is a remarkably difficult problem, as summarized in the famous paper “Nineteen dubious ways to compute the exponential matrix” [17]. However, computing the product of a matrix exponential and a vector, as in equation (1.2), can be done without explicitly computing  $\exp(At)$ . One idea is based on a contour integral

representation. The well-known Cauchy integral formula

$$f(z_0) = \frac{1}{2\pi i} \int_{\Gamma} \frac{f(z)}{(z - z_0)} dz,$$

evaluates the analytic function  $f$  via an integral along a closed contour  $\Gamma$  that encloses  $z_0$ . The matrix form of this formula is given by [14]

$$f(A) = \frac{1}{2\pi i} \int_{\Gamma} f(z) (zI - A)^{-1} dz,$$

where  $I$  is the identity matrix and the contour  $\Gamma$  encloses all the eigenvalues of  $A$ . If we define  $f$  by the analytic function

$$f(z) = e^{zt},$$

then from (1.2) we have

$$\mathbf{u}(t) = f(A)\mathbf{u}_0.$$

Hence, the contour integral representation of (1.2) is given by

$$\mathbf{u}(t) = \frac{1}{2\pi i} \int_{\Gamma} e^{zt} ((zI - A)^{-1}\mathbf{u}_0) dz. \quad (1.3)$$

To evaluate the integrand in (1.3) for numerical purposes, as we will see in Section 2.4, we need to solve linear systems of the form

$$(zI - A)\mathbf{x} = \mathbf{u}_0, \quad (1.4)$$

where  $A \in \mathbb{R}^{n \times n}$ ,  $I \in \mathbb{R}^{n \times n}$  and  $\mathbf{x}, \mathbf{u}_0 \in \mathbb{R}^n$ . This has to be solved for several values of the complex scalar  $z$  (each value of  $z$  corresponds to a node in the numerical integration rule for computing (1.3)). Such shifted linear systems also arise in many other science and engineering problems.

This thesis considers efficient methods for solving systems of the form (1.4). Here, we want to solve (1.4) for several values of  $z$ , while the matrix  $A$  and the right-hand side vector  $\mathbf{u}_0$  remain constant. One may wish to apply Gaussian elimination or, equivalently,  $LU$  factorization to (1.4), but this is expensive when many  $z$ 's and large systems are considered.

Note that in the related case, where the system

$$A\mathbf{x} = \mathbf{b},$$



has to be solved for several right-hand sides, an  $LU$  decomposition can be used to reduce the computational time. Since

$$LU\mathbf{x} = \mathbf{b},$$

letting  $\mathbf{y} = U\mathbf{x}$  we have

$$L\mathbf{y} = \mathbf{b}, \quad U\mathbf{x} = \mathbf{y},$$

and then each solution requires only forward and backward substitution. The same trick cannot, however, be used for the shifted systems (1.4) for several values of  $z$ .

## 1.2 Motivation

The motivation of this study is the work done in [11], where the authors pointed out that the shifted linear systems from the contour integral method can be solved efficiently using iterative methods with some preconditioning. When finite difference methods are used in the spatial discretization of the PDE, the resulting matrix  $A$  will be sparse and of large order. Applying Gaussian elimination to these systems does not take advantage of the sparsity of the matrix and this results in the problem of fill-in [7, p. 83–84], [20, p. 75]. The alternative to direct methods like Gaussian elimination is iterative methods. Many iterative methods like the Jacobi and Gauss-Seidel methods can be used, but none of them are efficient, as they cannot use information from one  $z$  for another  $z$ . On the other hand, because of their shift-invariance property, the Krylov subspace methods allow one to obtain approximate solutions of (1.4) by generating a single approximation space for all values of  $z$  [16, 20, 21].

The authors of [11] used the full orthogonalization method (FOM) with shift-invert preconditioning to solve the shifted systems that arise from the contour integral method. They found that the contour approximation method combined with this Krylov subspace method performs better than the other numerical methods they discussed. This was our motivation for this work. In this thesis, we combine the contour integral method not only with FOM but also with three other Krylov methods, the general minimal residual method (GMRES), the Lanczos method (LM) and the minimal residual method (MINRES).

On the other hand, when spectral methods are used instead of finite differences for the spatial discretization, the matrix  $A$  will be dense but of relatively small order. As we explained

above, applying Gaussian elimination to the shifted systems for each  $z$  is expensive. Here we show how the Hessenberg decomposition can be used to reduce the system to almost upper triangular form, which can then be solved efficiently.

Therefore, depending on the space discretization applied to the PDE, the contour integral approximation results in shifted linear systems with either a sparse or a dense matrix. This thesis discusses efficient methods for solving shifted linear systems with both dense and sparse matrices and compares them with built-in direct linear system solvers.

The sparse solver that we use for comparison in this thesis is based on `UMFPACK` [5], and in particular we use `spsolve` from the SciPy library [4]. The algorithm is based on an effective pre-ordering strategy that aims to minimize the fill-in when the  $LU$  factorization is computed [6, p. 138].

All numerical results and computational times listed in this thesis were computed using SciPy version 0.7.0 in Python 2.6 on a machine running the Ubuntu Linux operating system. The central processing unit (CPU) was an Intel Xeon running at 2000MHZ with 16GB memory. Although parallel or multicore implementation would have improved the execution speed of many of our algorithms, we have not pursued this and we have done all computations using a single processor.

### 1.3 Organization of the thesis

In Chapter 2 we present the three PDEs that we have selected as test problems. Firstly, we give the analytical solutions, which can be used for computing the errors in the numerical methods, and discuss the numerical methods for solving these model problems. Secondly, we introduce space discretization methods, namely finite differences and Chebyshev collocation methods. Thirdly, we discuss the contour integral approximation method for treating the time variable.

In Chapter 3 we discuss an efficient method based on Hessenberg decomposition for solving shifted linear systems with a dense matrix. Firstly, we review the Hessenberg reduction using Householder reflections in view of implementation aspects. We found that the built-in `hessenberg` function in SciPy for the explicit computation of the orthogonal matrix is inefficient and we implement it more efficiently here. We compare the improved function

with the built-in function. Secondly, we introduce the Hessenberg reduction method for solving dense shifted systems. Thirdly, we compare the CPU-time of the built-in direct linear system solver (DLSS) and the Hessenberg reduction method (HRM) when solving shifted systems.

In Chapter 4 we discuss efficient methods for solving shifted systems with a sparse matrix. Firstly, we present the Krylov subspace methods. We discuss two methods for constructing an orthogonal basis of the Krylov subspace, namely the Arnoldi and Lanczos processes when  $A$  is nonsymmetric and symmetric, respectively. In addition, we introduce Krylov methods, namely FOM, GMRES, LM and MINRES, with orthogonality and minimization constraints to approximate the solution of a linear system. Secondly, we show how these Krylov methods are extended to solve the shifted systems. Thirdly, since the Krylov methods suffer from slow convergence, we also discuss preconditioning to improve convergence. Fourthly, we compare the Krylov methods and the built-in direct sparse solver (DSS) numerically and give a conclusion.

# Chapter 2

## Benchmark Problems

### 2.1 Introduction

In this chapter we present benchmark problems with their analytical solutions and discuss the numerical methods for solving these problems.

This chapter is organized as follows. In Section 2.2 we present three model problems involving the heat equation and their analytical solutions, which one can use as reference for computing the total error in the numerical methods. In Section 2.3 we discuss two space discretization methods, namely the finite difference method [12] and the spectral method [3, 22]. The finite difference method results in a system of ODEs in time with sparse matrix of large order. On the other hand, the spectral method results in a system with a dense matrix. Finally, in Section 2.4 we introduce the contour integral approximation method to advance the solution in time.

### 2.2 Heat Equations

#### 2.2.1 The 1D Heat Equation

Our first model problem is the one dimensional heat equation [8, p. 399]

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < 1 \text{ and } t > 0, \quad (2.1)$$

where  $\kappa$  is a positive constant, and initial and boundary conditions are given by

$$\begin{aligned} u(x, 0) &= 0, & 0 < x < 1, \\ u(0, t) &= 0, & u(1, t) = 1, & t > 0. \end{aligned}$$

Since homogeneous boundary conditions make finding the solution easier, we use the change of variable  $v = u - x$ , which results in

$$\frac{\partial v}{\partial t} = \kappa \frac{\partial^2 v}{\partial x^2}, \quad (2.2)$$

with initial condition

$$v(x, 0) = -x, \quad 0 < x < 1,$$

and homogeneous boundary conditions

$$v(0, t) = 0, \quad v(1, t) = 0, \quad t > 0.$$

The analytical solution to this model problem is given by [8, p. 400]

$$u(x, t) = x + \frac{2}{\pi} \sum_{n=1}^{\infty} \frac{(-1)^n}{n} \sin(n\pi x) e^{-\kappa t(n\pi)^2}. \quad (2.3)$$

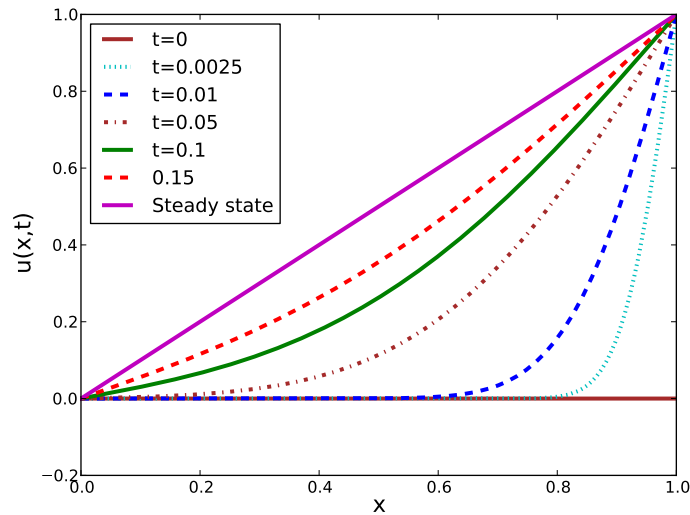


Figure 2.1. Analytical solution of the one-dimensional heat equation (2.1) at different times and the steady state solution for  $\kappa = 1$ .

## 2.2.2 The 2D Heat Equation

The second model problem that we consider is the two dimensional heat equation [19, p. 953]

$$\frac{\partial u}{\partial t} = \kappa \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right), \quad 0 < x < 1, \quad 0 < y < 1 \text{ and } t > 0, \quad (2.4)$$

with initial and boundary conditions given by

$$\begin{aligned} u(x, 0, t) &= u(x, 1, t) = 0, \quad 0 < x < 1, \quad t > 0, \\ u(0, y, t) &= u(1, y, t) = 0, \quad 0 < y < 1, \quad t > 0, \\ u(x, y, 0) &= x(1 - x^2)y(1 - y). \end{aligned}$$

This problem has a series solution given by [19, p. 954]

$$u(x, y, t) = \frac{48}{\pi^6} \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} \frac{(-1)^n ((-1)^m - 1)}{n^3 m^3} \sin(n\pi x) \sin(m\pi y) e^{-(n^2+m^2)\pi^2 \kappa t}.$$

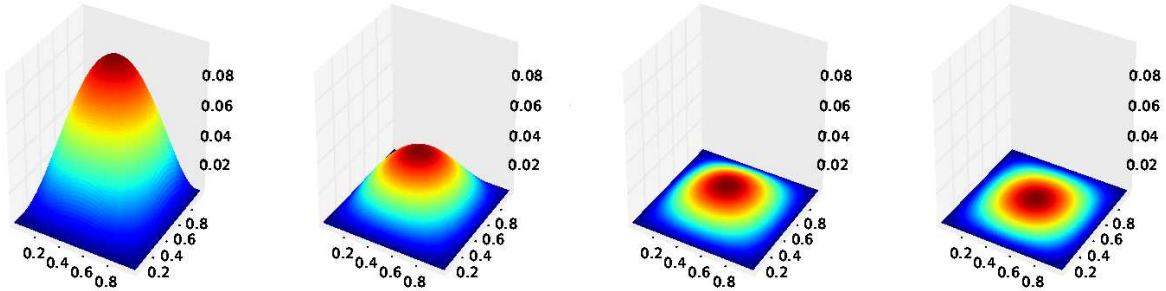


Figure 2.2. Analytical solution of the two-dimensional heat equation (2.4) at time  $t = 0$ ,  $t = 0.05$ ,  $t = 0.1$  and  $t = 0.15$  from left to right for  $\kappa = 1$ .

The other model problem that we consider is obtained from the above problem by adding a convection term, namely

$$\frac{\partial u}{\partial t} = \kappa \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + c \frac{\partial u}{\partial x}, \quad 0 < x < 1, \quad 0 < y < 1 \text{ and } t > 0, \quad (2.5)$$

with the same initial and boundary conditions. We solve this model problem by the method

of separation of variables and obtain the series solution

$$u(x, y, t) = e^{-\left(\frac{c}{2\kappa}\right)x} \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} c_{nm} \sin(n\pi x) \sin(m\pi y) e^{-\left(n^2\pi^2 + m^2\pi^2 + \left(\frac{c}{2\kappa}\right)^2\right)\kappa t},$$

where

$$c_{nm} = \frac{128}{\pi^3} \left( \frac{(-1)^m - 1}{m^3} \right) \left[ \left( 2(-1)^n e^{\left(\frac{c}{2\kappa}\right)} \left( \frac{16(\pi^5 c \kappa^7 + 3\pi^5 \kappa^8) n^5 + 8(\pi^3 c^3 \kappa^5 - 3\pi^3 c^2 \kappa^6 - 12\pi^3 c \kappa^7) n^3}{(4\pi^2 \kappa^2 n^2 + c^2)^4} \right. \right. \right. \\ \left. \left. \left. + \frac{(\pi c^5 \kappa^3 - 9\pi c^4 \kappa^4 + 24\pi c^3 \kappa^5) n}{(4\pi^2 \kappa^2 n^2 + c^2)^4} \right) \right) \right. \\ \left. \left. + \left( \frac{(8(2\kappa^2(\pi^2 n^2) + c^2)\pi^2 \kappa^2 n^2 + c^4 - 48c^2 \kappa^2)\pi c \kappa^3 n}{(4\pi^2 \kappa^2 n^2 + c^2)^4} \right) \right] \right]. \quad (2.6)$$

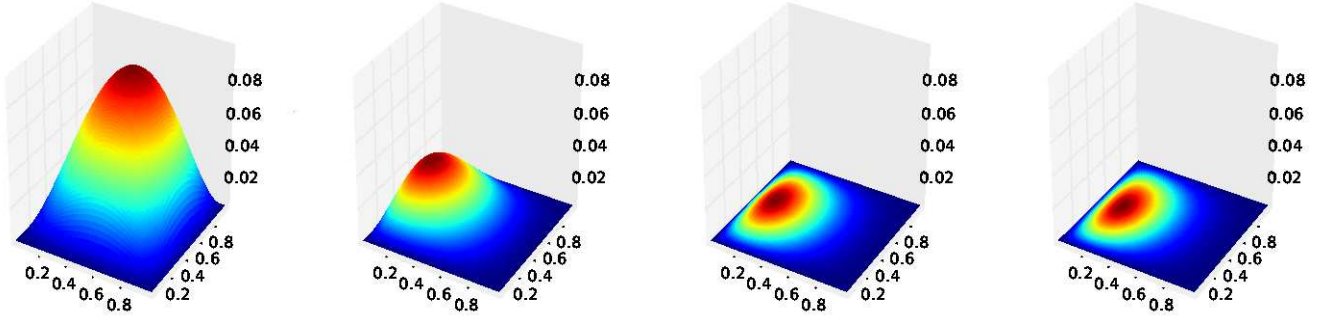


Figure 2.3. Analytical solution of the two-dimensional heat equation with convection term at time  $t = 0.0$ ,  $t = 0.05$ ,  $t = 0.1$  and  $t = 0.15$  from left to right for  $c = 10.0$  and  $\kappa = 1$ .

## 2.3 Space Discretization Methods

Space discretization methods approximate the spatial derivatives of a continuous function by a function of finite grid values; as a result, we obtain a finite system of equations to be solved. The simplest of these space discretization methods is the finite difference method, which we discuss next.

### 2.3.1 Finite Differences

To solve the problem numerically, we start by discretizing the spatial domain into uniformly spaced grid points. The finite difference method approximates the derivatives of the function by the derivatives of the local interpolant on the grid [22, p. 2]. Before we discretize the two dimensional problem let us see the discretization of a function of one variable  $u(x)$  on an interval  $[a, b]$ .

#### One Dimensional Finite Difference Formulas

The finite difference approximation of the first derivative of  $u(x)$  at the mesh point  $x_i$  is given by [18, p. 149]

$$u'(x_i) = \frac{u(x_{i+1}) - u(x_{i-1}))}{2h} + O(h^2),$$

where  $h$  is the step size. This is called the central difference formula, which we implement as

$$u'_i = \frac{u_{i+1} - u_{i-1}}{2h}.$$

Likewise, the second derivative central difference approximation is given by [7, p. 267]

$$u''_i = \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}. \quad (2.7)$$

#### Two Dimensional Finite Difference Formulas

The central difference formulas in two dimensions are found by discretizing the rectangular spatial domain  $[a, b] \times [c, d]$  of the problem. Let  $h_1 = \frac{b-a}{N+1}$  be the step-size in the  $x$ -direction and  $h_2 = \frac{d-c}{M+1}$  be the step-size in the  $y$ -direction, then the uniform rectangular grid points are

$$x_i = a + ih_1, \quad i = 0, 1, \dots, N + 1,$$

and

$$y_j = c + jh_2, \quad j = 0, 1, \dots, M + 1.$$

Let  $u_{i,j}$  be the approximation of the value  $u(x, y, t)$  at the point  $(x_i, y_j)$ .



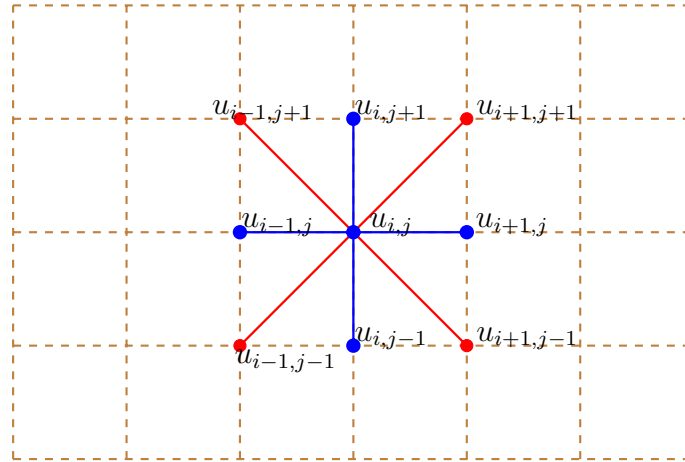


Figure 2.4. A 2D grid with grid function values at its mesh points. The grid function values  $u_{i,j}$  and four of its neighbours, vertical and horizontal, are used in the 5-point central difference formula (2.8). In addition to these five points the other diagonal points are used in the 9-point central difference formula (2.10).

Then the second-order partial derivatives at  $(x_i, y_j)$  can be found using (2.7) as [7, p. 270]

$$\frac{\partial^2 u(x_i, y_j)}{\partial x^2} \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_1^2},$$

and

$$\frac{\partial^2 u(x_i, y_j)}{\partial y^2} \approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_2^2}.$$

When we use a square domain with the same step size  $h$  in both directions, i.e.,  $h_1 = h_2$ , and  $N = M$  we have [7, p. 271]

$$\frac{\partial^2 u(x_i, y_j)}{\partial x^2} + \frac{\partial^2 u(x_i, y_j)}{\partial y^2} \approx \frac{u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i,j+1} + u_{i+1,j}}{h^2}, \quad 1 \leq i, j \leq N, \quad (2.8)$$

which is called the 5-point central difference approximation of the Laplacian. We can now approximate the model problem (2.4) using this formula. By ordering the unknowns

$\{u_{i,j}\}_{i,j=1}^N$  in the natural ordering as

$$\mathbf{u}(t) = \begin{pmatrix} u_{1,1} \\ u_{2,1} \\ \vdots \\ u_{N,1} \\ \text{---} \\ u_{1,2} \\ u_{2,2} \\ \vdots \\ u_{N,2} \\ \text{---} \\ \vdots \\ \text{---} \\ u_{1,N} \\ u_{2,N} \\ \vdots \\ u_{N,N} \end{pmatrix},$$

the model problem (2.4) can be written as

$$\frac{d\mathbf{u}}{dt} = \frac{\kappa}{h^2} A_1 \mathbf{u}(t), \quad \mathbf{u}(0) = \mathbf{u}_0, \quad (2.9)$$

where

$$A_1 = \begin{pmatrix} B_1 & I & 0 & & 0 \\ I & B_1 & I & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ & & & & I \\ 0 & \dots & & I & B_1 \end{pmatrix}.$$

In the block tridiagonal matrix  $A_1$ ,  $I \in \mathbb{R}^{N \times N}$  is the identity matrix and  $B_1 \in \mathbb{R}^{N \times N}$  is

the tridiagonal matrix given by

$$B_1 = \begin{pmatrix} -4 & 1 & 0 & & 0 \\ 1 & -4 & 1 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ & & & & 1 \\ 0 & \dots & & 1 & -4 \end{pmatrix}.$$

The other finite difference formula is the 9-point difference approximation which gives a better accuracy than the 5-point approximation. The 9-point formula is a linear combination of all the nine points, horizontal, vertical and diagonal, in Figure 2.1,

$$\begin{aligned} \frac{\partial^2 u(x_i, y_j)}{\partial x^2} + \frac{\partial^2 u(x_i, y_j)}{\partial y^2} &\approx \beta_{-1,-1}u_{i-1,j-1} + \beta_{0,-1}u_{i,j-1} + \beta_{1,-1}u_{i+1,j-1} + \beta_{-1,0}u_{i-1,j} \\ &+ \beta_{0,0}u_{i,j} + \beta_{1,0}u_{i+1,j} + \beta_{-1,1}u_{i-1,j+1} + \beta_{0,1}u_{i,j+1} + \beta_{1,1}u_{i+1,j+1}, \end{aligned}$$

where  $1 \leq i, j \leq N$ . To determine the  $\beta$ 's we need to subtract the derivatives that we want to approximate from the Taylor expansion of  $u$  and we require that all the terms up to some error or order  $h^p$  vanish. However, in this case the requirement that the error be of order  $h^2$  will not be enough to fix the  $\beta$ 's uniquely, and if we go up to  $h^3$  there will be no solution. So, the standard 9-point central difference formula is defined by the choice [12, p. 159]

$$\beta_{i,j} = \begin{cases} -10/3h^2, & \text{if } i = j = 0, \\ 2/3h^2, & \text{if } |i + j| = 1, \\ 1/6h^2, & \text{otherwise,} \end{cases}$$

which gives

$$\begin{aligned} \frac{\partial^2 u(x_i, y_j)}{\partial x^2} + \frac{\partial^2 u(x_i, y_j)}{\partial y^2} &\approx \frac{u_{i-1,j-1} + 4u_{i,j-1} + u_{i+1,j-1} + 4u_{i-1,j}}{6h^2} \\ &+ \frac{-20u_{i,j} + 4u_{i+1,j} + u_{i-1,j+1} + 4u_{i,j+1} + u_{i+1,j+1}}{6h^2}, \quad 1 \leq i, j \leq N. \end{aligned} \quad (2.10)$$

This is called the 9-point approximation to the Laplacian. Thus, this approximation with the natural ordering changes the model problem (2.4) to a system of ODEs

$$\frac{d\mathbf{u}}{dt} = \frac{\kappa}{6h^2} A_2 \mathbf{u}(t), \quad \mathbf{u}(0) = \mathbf{u}_0, \quad (2.11)$$

where

$$A_2 = \begin{pmatrix} B_2 & C_2 & 0 & & 0 \\ C_2 & B_2 & C_2 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ & & & & C_2 \\ 0 & \dots & & C_2 & B_2 \end{pmatrix}.$$

The tridiagonal matrices  $B_2, C_2 \in \mathbb{R}^{N \times N}$  in  $A_2$  are given by

$$B_2 = \begin{pmatrix} -20 & 4 & 0 & & 0 \\ 4 & -20 & 4 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ & & & & 4 \\ 0 & \dots & & 4 & -20 \end{pmatrix},$$

and

$$C_2 = \begin{pmatrix} 4 & 1 & 0 & & 0 \\ 1 & 4 & 1 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ & & & & 1 \\ 0 & \dots & & 1 & 4 \end{pmatrix}.$$

By absorbing the constant coefficients  $\frac{\kappa}{h^2}$  into  $A_1$  and  $\frac{\kappa}{6h^2}$  into  $A_2$  in (2.9) and (2.11), respectively, we obtain a system of ODEs of the form

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u}(t), \quad \mathbf{u}(0) = \mathbf{u}_0, \quad (2.12)$$

where  $A$  is a symmetric sparse matrix of large order.

We also approximate the model problem (2.5) by the finite differences as

$$\begin{aligned} \frac{\partial u(x_i, y_j)}{\partial t} \approx \kappa & \left( \frac{u_{i-1,j-1} + 4u_{i,j-1} + u_{i+1,j-1} + 4u_{i-1,j} - 20u_{i,j} + 4u_{i+1,j} + u_{i-1,j+1}}{6h^2} \right. \\ & \left. + \frac{4u_{i,j+1} + u_{i+1,j+1}}{6h^2} \right) + c \frac{u_{i+1,j} - u_{i-1,j}}{2h}, \quad 1 \leq i, j \leq N. \end{aligned} \quad (2.13)$$

This finite difference approximation with the natural ordering results in a system of ODEs

of the form (2.12) where

$$A = \begin{pmatrix} B & C & 0 & & 0 \\ C & B & C & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ & & & & C \\ 0 & \dots & & C & B \end{pmatrix}.$$

The tridiagonal matrices  $B, C \in \mathbb{R}^{N \times N}$  in  $A$  are given by

$$B = \frac{1}{6h^2} \begin{pmatrix} -20\kappa & 4\kappa + 3hc & 0 & & 0 \\ 4\kappa - 3hc & -20\kappa & 4\kappa + 3hc & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ & & & & 4\kappa + 3hc \\ 0 & \dots & & 4\kappa - 3hc & -20\kappa \end{pmatrix},$$

and

$$C = \frac{\kappa}{6h^2} \begin{pmatrix} 4 & 1 & 0 & & 0 \\ 1 & 4 & 1 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ & & & & 1 \\ 0 & \dots & & 1 & 4 \end{pmatrix},$$

Here the matrix  $A$  is nonsymmetric unlike the symmetric matrix in the second model problem. In both model problems the finite difference methods change the PDEs into systems of ODEs (2.12) in time with a sparse matrix  $A$  of large order. Next we discuss another space discretization method called the Chebyshev collocation method which results in a system with a dense matrix but typically smaller.

### 2.3.2 Chebyshev Collocation Spectral Method

Spectral methods are global methods, i.e., the value of the derivative at a certain point in space depends on the solution at all the other points in space, and not just the neighbouring grid points like finite difference methods. This means that matrices are full rather than sparse, but because of the superior accuracy of global interpolants the matrices are also smaller.

In the Chebyshev collocation method the canonical interval is  $[-1, 1]$ , so we use the trans-

formation  $x = \frac{1}{2}(s + 1)$  and the model problem (2.2) becomes

$$\frac{\partial v}{\partial t} = 4\kappa \frac{\partial^2 v}{\partial s^2}, \quad -1 < s < 1, \quad t > 0, \quad (2.14)$$

with initial and boundary conditions

$$v(s, 0) = -\frac{1}{2}(s + 1), \quad -1 < s < 1,$$

$$v(-1, t) = 0, \quad v(1, t) = 0, \quad t > 0.$$

The most convenient and commonly used collocation points for this problem are the Chebyshev points of the second kind [3, p. 13][25]

$$s_k = \cos\left(\frac{\pi k}{N}\right), \quad k = 0, 1, \dots, N,$$

which are extrema of the  $N^{\text{th}}$  order Chebyshev polynomial defined on  $[-1, 1]$  by

$$T_N(s) = \cos(N \cos^{-1} s).$$

Thus, we seek a solution to (2.14) of the form

$$v_N(s, t) = \sum_{k=0}^N v_N(s_k, t) \phi_k(s), \quad (2.15)$$

where  $v_N(s, t)$  is a polynomial of degree  $N$  and  $\phi_k(s)$  is the interpolating Lagrange polynomial given by [3, p. 88]

$$\phi_k(s) = \frac{(-1)^{k+1}(1-s^2)T'_N(s)}{c_k N^2(s-s_k)}, \quad (2.16)$$

and

$$c_k = \begin{cases} 2 & k = 0, N \\ 1 & 1 \leq k \leq N - 1. \end{cases}$$

In the collocation method the requirement is that (2.15) has to satisfy (2.14) exactly at the set of collocation points  $s_k$  in  $[-1, 1]$ . Thus, we have

$$\frac{\partial v_N(s_k, t)}{\partial t} = 4\kappa \frac{\partial^2 v_N(s_k, t)}{\partial s^2}, \quad k = 1, \dots, N - 1,$$

with initial and boundary conditions

$$\begin{aligned} v_N(s_k, 0) &= -\frac{1}{2}(s_k + 1), \quad k = 1, \dots, N-1, \\ v_N(-1, t) &= 0, \quad v_N(1, t) = 0, \quad t > 0. \end{aligned}$$

The Chebyshev interpolation derivative can be represented in a matrix form as

$$\frac{dv_N(s_k, t)}{dt} = 4\kappa \sum_{j=0}^N D_{k,j}^{(2)} v_N(s_k, t), \quad k = 1, \dots, N-1, \quad (2.17)$$

where the entries  $D_{k,j}^{(2)}$  are obtained from the differentiation of the characteristic Lagrange polynomial (2.16). The entries of the first differentiation matrix  $D^{(1)} = [D_{k,j}^{(1)}]$  are given by [3, p. 89]

$$D_{k,j}^{(1)} = \begin{cases} \frac{c_k}{c_j} \frac{(-1)^{k+j}}{(x_k - x_j)} & k \neq j \\ -\frac{x_k}{2(1-x_k^2)} & 1 \leq k = j \leq N-1 \\ \frac{2N^2+1}{6} & k = j = 0 \\ -\frac{2N^2+1}{6} & k = j = N, \end{cases}$$

and

$$D^{(\ell)} = (D^{(1)})^\ell, \quad \ell = 1, 2, \dots$$

Note that the computation of  $D^{(2)}$  in (2.17) using the first derivative matrix  $D^{(1)}$  is sensitive to round-off errors and computationally expensive. Therefore, it is recommended for practical computations to use the recurrence version of the above formula given in [25].

Let  $\mathbf{v}(t) = (v_N(s_1, t), \dots, v_{N-1}(s_{N-1}, t))^T$  and  $\mathbf{s} = (s_1, s_2, \dots, s_{N-1})^T$ , then (2.17) can be written as

$$\frac{d\mathbf{v}}{dt} = A\mathbf{v}(t), \quad \mathbf{v}(0) = -\frac{1}{2}(\mathbf{s} - \mathbf{1}), \quad (2.18)$$

where  $A = 4\kappa \tilde{D}^{(2)}$  and  $\tilde{D}^{(2)}$  is obtained from  $D^{(2)}$  by deleting the first and last rows and first and last columns. The differentiation matrix  $D^{(2)}$  is dense and so is the matrix  $A$ . Even though  $A$  is dense, it is typically much smaller than the corresponding finite difference matrix  $A$ , because global interpolating polynomials are more accurate than local ones.

In the next section we discuss methods for the time integration of the systems of ODEs (2.12) and (2.18).

## 2.4 Time Integration Methods

Both spatial discretization methods, namely spectral and finite difference methods, result in systems of ODEs of the form (2.12) where  $\mathbf{u}$  is the vector containing the unknown solution. For the time integration of (2.12) we adopt a relatively new approach using a contour approximation method as discussed in [26].

### 2.4.1 Contour Integral Method

Two contours, a parabola and a hyperbola, have been discussed in [26] for the computation of (1.3). In this thesis we use the parabolic contour which is parametrized by

$$z = \mu(i\phi + 1)^2, \quad -\infty < \phi < \infty,$$

where  $\mu$  is a positive parameter which controls the width of the contour.

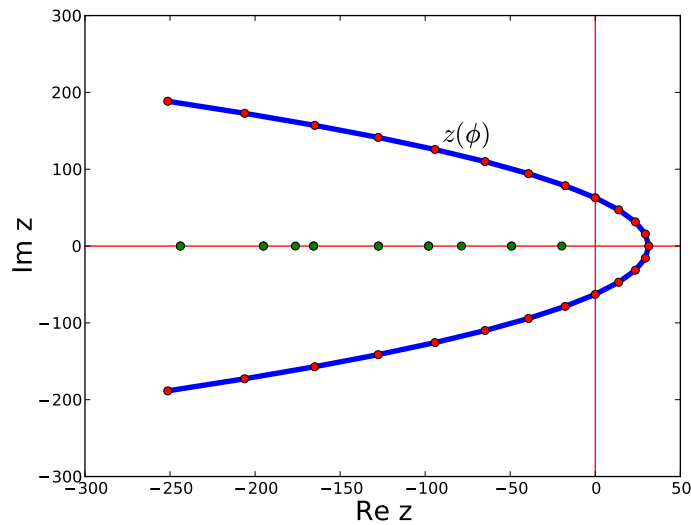


Figure 2.5. Schematic plot of the parabolic contour. The nodal points  $z_k$  on the contour  $z(\phi)$  are defined in (2.22) below. The dots on the real axis are some of the eigenvalues of the matrix  $A$ , which indicates that our contour encloses all the eigenvalues.



Thus, the contour integral (1.3) becomes

$$\mathbf{u}(t) = \frac{1}{2\pi i} \int_{-\infty}^{\infty} e^{z(\phi)t} \mathbf{U}(z(\phi)) z'(\phi) d\phi, \quad (2.19)$$

where

$$(z(\phi)I - A)\mathbf{U}(z(\phi)) = \mathbf{u}_0. \quad (2.20)$$

This integral can be evaluated by numerical integration methods, for example the trapezoidal rule, at a cost of solving the shifted linear system (2.20) at each node of the contour.

## 2.4.2 The Trapezoidal Rule

The trapezoidal rule approximation of the integral (2.19) with uniform node spacing  $h$  is given by

$$\mathbf{u}(t) \approx \frac{h}{2\pi i} \sum_{k=-\infty}^{\infty} e^{z_k t} z'_k \mathbf{U}_k, \quad (2.21)$$

where

$$\phi_k = kh, \quad z_k = z(\phi_k), \quad z'_k = z'(\phi_k), \quad \mathbf{U}_k = \mathbf{U}(z_k). \quad (2.22)$$

In a practical computation we need to truncate the infinite series (2.21). The finite approximation  $\mathbf{u}_{h;M}(t)$  of the infinite series (2.21) is then given by

$$\mathbf{u}_{h;M}(t) = \frac{h}{2\pi i} \sum_{k=-M}^M e^{z_k t} z'_k \mathbf{U}_k, \quad (2.23)$$

where

$$(z_k I - A)\mathbf{U}_k = \mathbf{u}_0. \quad (2.24)$$

Each term in the sum (2.23) involves solving the shifted linear systems (2.24). Thus, the major computational cost is solving the systems (2.24) for each  $z_k$ . Since the parabolic contour is symmetric with respect to the real axis, the quadrature sum (2.23) can be approximated by

$$\mathbf{u}_{h;M}(t) = \frac{h}{\pi} \operatorname{Im} \left\{ \frac{1}{2} e^{z_0 t} z'_0 \mathbf{U}_0 + \sum_{k=1}^M e^{z_k t} z'_k \mathbf{U}_k \right\}. \quad (2.25)$$

This symmetry reduces the number of linear systems to be solved by half. The optimal  $h$  and  $\mu$  for the trapezoidal rule, when the matrix  $A$  has real, negative eigenvalues, are given

by [26]

$$h = \frac{3}{M}, \quad \mu = \frac{\pi M}{12t}, \quad (2.26)$$

where  $M$  is number of nodes on the contour.

In the next chapters we use the formula (2.25) to solve the three model problems introduced earlier in this chapter.

# Chapter 3

## Dense shifted linear systems

### 3.1 Introduction

As we discussed in the previous chapter, the spectral method followed by the contour integral method results in shifted linear systems (2.24) with a dense matrix  $A$ . In this chapter we present a method to solve these systems efficiently. When we apply the direct linear system solver (DLSS) to the systems (2.24) we effectively have to apply Gaussian elimination to the shifted matrix  $(z_k I - A)$  for each  $z_k$ . This is computationally expensive and applying a prior  $LU$  factorization cannot help either. However, as we will see later in this chapter only one Hessenberg reduction of the matrix  $A$  is needed to reduce the dense system into an almost upper triangular system for all  $z_k$  and the resulting systems are then efficiently solved by Gaussian elimination, since only one element in each column has to be eliminated.

This chapter is organized as follows. In Section 3.2 we discuss the Hessenberg reduction via Householder reflections. When we used the built-in `hessenberg` function in SciPy we found that the computation of the orthogonal matrix in the Hessenberg decomposition is implemented inefficiently. That is why we discuss the Hessenberg reduction here. We improved this function to make it more efficient, so that its execution time agrees with the theoretical operation count. In Section 3.3 we introduce the Hessenberg reduction method. In Section 3.4 we present the numerical comparison of the Hessenberg reduction method and the direct linear system solvers. Finally, we give a summary.

## 3.2 The Hessenberg Reduction

Many problems in science and engineering result in linear algebraic problems that involve very large matrices  $A \in \mathbb{R}^{n \times n}$ . The first approach to solving such problems is to reduce the matrix  $A$  into another matrix that requires less storage or is better suited for further processing than  $A$ . Hessenberg reduction, which is based on orthogonal transformation, is one of the most useful methods of reducing a matrix. First, we introduce the Householder reflections and later we apply them to compute the Hessenberg reduction.

### 3.2.1 Householder Reflections

Let  $\mathbf{v} \in \mathbb{R}^n$  be nonzero. A matrix  $P \in \mathbb{R}^{n \times n}$  of the form

$$P = I - \frac{2}{\mathbf{v}^T \mathbf{v}} \mathbf{v} \mathbf{v}^T, \quad (3.1)$$

is called a *Householder reflection* or *Householder matrix* and  $\mathbf{v}$  is called a *Householder vector* [9, p. 209]. Householder matrices are symmetric and orthogonal, since

$$P^T = I^T - \frac{2}{\mathbf{v}^T \mathbf{v}} (\mathbf{v}^T)^T \mathbf{v}^T = P,$$

and

$$P P^T = I - \frac{4}{\mathbf{v}^T \mathbf{v}} \mathbf{v} \mathbf{v}^T + \frac{4}{\mathbf{v}^T \mathbf{v}} \mathbf{v} \mathbf{v}^T = I.$$

The Householder reflection can be used to replace specified entries in a vector by zero. Given  $\mathbf{0} \neq \mathbf{x} \in \mathbb{R}^n$ , then there exists a Householder reflection that can replace all but the first entry in  $\mathbf{x}$  by zero, such that  $P\mathbf{x}$  is a multiple of  $\mathbf{e}_1 = (1, 0, \dots, 0)^T$ , the first unit vector. Note that

$$P\mathbf{x} = \mathbf{x} - \frac{2\mathbf{v}^T \mathbf{x}}{\mathbf{v}^T \mathbf{v}} \mathbf{v},$$

and  $P\mathbf{x} \in \text{span}\{\mathbf{e}_1\}$  implies  $\mathbf{v} \in \text{span}\{\mathbf{x}, \mathbf{e}_1\}$ . Setting  $\mathbf{v} = \mathbf{x} + \alpha \mathbf{e}_1$  gives

$$P\mathbf{x} = \left( 1 - 2 \frac{\mathbf{x}^T \mathbf{x} + \alpha x_1}{\mathbf{x}^T \mathbf{x} + 2\alpha x_1 + \alpha^2} \right) \mathbf{x} - 2\alpha \frac{\mathbf{v}^T \mathbf{x}}{\mathbf{v}^T \mathbf{v}} \mathbf{e}_1.$$

Since  $P\mathbf{x}$  is a multiple of  $\mathbf{e}_1$ , the coefficient of  $\mathbf{x}$  has to be zero which gives

$$\alpha = \pm \|\mathbf{x}\|_2,$$

where

$$\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^T \mathbf{x}}.$$

Then

$$\mathbf{v} = \mathbf{x} \pm \|\mathbf{x}\|_2 \mathbf{e}_1,$$

which implies that

$$P\mathbf{x} = \mp \|\mathbf{x}\|_2 \mathbf{e}_1. \quad (3.2)$$

Although both mappings are satisfactory in theory, for a given  $\mathbf{x}$ , one will have better numerical stability properties than the other. Setting  $v_1 = x_1 - \|\mathbf{x}\|_2$  is dangerous if  $\mathbf{x}$  is close to a positive multiple of  $\mathbf{e}_1$  because severe cancellation could occur when we subtract two nearly equal numbers. However, the formula

$$\begin{aligned} v_1 = x_1 - \|\mathbf{x}\|_2 &= \frac{x_1^2 - \|\mathbf{x}\|_2^2}{x_1 + \|\mathbf{x}\|_2} \\ &= \frac{-(x_2^2 + \cdots + x_n^2)}{x_1 + \|\mathbf{x}\|_2}, \end{aligned}$$

does not suffer from this defect in the  $x_1 > 0$  case. In other words, we want  $\pm \|\mathbf{x}\|_2 \mathbf{e}_1$  not too close to  $\mathbf{x}$ . To achieve this, we can choose

$$\mathbf{v} = \mathbf{x} + \text{sign}(x_1) \|\mathbf{x}\|_2 \mathbf{e}_1,$$

where

$$\text{sign}(x) = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0. \end{cases}$$

In practice, it is better to normalize the Householder vector so that  $v_1 = 1$ . Now, letting  $\beta = 2/(\mathbf{v}^T \mathbf{v})$  and  $n = \text{length}(\mathbf{x})$ , Algorithm 3.1 computes  $\beta$  and the Householder vector  $\mathbf{v}$  so that the Householder matrix is given by

$$P = I - \beta \mathbf{v} \mathbf{v}^T. \quad (3.3)$$

### 3.2.2 The Hessenberg Reduction via Householder Reflections

Given  $A \in \mathbb{R}^{n \times n}$ , the Hessenberg decomposition of  $A$  is given by

$$A = QHQ^T, \quad (3.4)$$

---

**Algorithm 3.1** Computation of Householder vector  $[\mathbf{v}, \beta] = \text{house}(\mathbf{x})$  [9, p. 201].

---

```

1:  $\sigma = \mathbf{x}(2:n)^T \mathbf{x}(2:n)$ 
2:  $\mathbf{v} = [1 \quad \mathbf{x}(2:n)]^T$ 
3: if  $\sigma = 0$  then
4:    $\beta = 0$ 
5: else
6:    $\mu = \sqrt{x_1^2 + \sigma}$ 
7:   if  $x_1 \leq 0$  then
8:      $v_1 = x_1 - \mu$ 
9:   else
10:     $v_1 = -\sigma / (x_1 + \mu)$ 
11:   end if
12:    $\beta = 2v_1^2 / (\sigma + v_1^2)$ 
13:    $\mathbf{v} = \mathbf{v} / v_1$ 
14: end if
    
```

---

where  $Q$  is orthogonal and  $H$  is upper Hessenberg, i.e.,  $h_{ij} = 0$  whenever  $i > j + 1$ . The idea of Hessenberg decomposition is to apply orthogonal similarity transformations to  $A$  so as to introduce zeros below the first subdiagonal of  $A$ . This can be done by applying Householder reflections to each column of the matrix.

For the first step, we construct a Householder reflector  $Q_1$  such that multiplying  $A$  from the left by  $Q_1^T$  leaves the first row unaltered and sets all entries in the first column to zero below the first subdiagonal. Now, to complete the similarity transformation we need to multiply  $Q_1^T A$  from the right by  $Q_1$ ; note that the first column will be unchanged.

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \end{bmatrix} \xrightarrow{Q_1^T} \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \end{bmatrix} \xrightarrow{Q_1} \begin{bmatrix} \times & * & * & * & * & * \\ \times & * & * & * & * & * \\ & * & * & * & * & * \\ & * & * & * & * & * \\ & * & * & * & * & * \\ & * & * & * & * & * \end{bmatrix}.$$

Here,  $\times$  stands for an unchanged entry and  $*$  for a changed entry when the Householder reflector is applied. Similarly, in the second step we construct a Householder reflector  $Q_2$  which leaves the first 2 rows unaltered and set all entries in the second column below the third row to zero. To complete the similarity transformation multiply it by  $Q_2$  from the

right.

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \end{bmatrix} \xrightarrow{Q_2^T} \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & * & * & * & * \end{bmatrix} \xrightarrow{Q_2} \begin{bmatrix} \times & \times & * & * & * & * \\ \times & \times & * & * & * & * \\ & \times & * & * & * & * \\ & & * & * & * & * \\ & & * & * & * & * \\ & & * & * & * & * \end{bmatrix}.$$

Repeating this  $n - 2$  times,  $A$  is reduced to Hessenberg form

$$H = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix},$$

where

$$H = \underbrace{Q_{n-2}^T \dots Q_2^T Q_1^T}_{Q^T} A \underbrace{Q_1 Q_2 \dots Q_{n-2}}_Q = Q^T A Q,$$

and

$$Q_k = I - \beta \mathbf{v}_k \mathbf{v}_k^T, \quad k = 1, 2, \dots, n - 2.$$

The Householder vector at step  $k$  is given by  $\mathbf{v}_k = [0, \dots, 1, A_{k-1}(k+1 : n, k)]^T$  where  $A_{k-1}$  is the matrix obtained after  $k - 1$  orthogonal similarity transformations. The orthogonal matrix  $Q$  is given as a product of the Householder reflectors  $Q_k$

$$Q = Q_1 Q_2 \dots Q_{n-2}. \tag{3.5}$$

Given  $A \in \mathbb{R}^{n \times n}$ , Algorithm 3.2 overwrites  $A$  with  $H = Q^T A Q$  where  $H$  is upper Hessenberg and  $Q$  is the product of Householder matrices.

This algorithm takes an array that contains  $A \in \mathbb{R}^{n \times n}$  as input and returns an upper Hessenberg matrix  $H$ , whose nonzero elements are in their natural position in the array. The portion of the array below the subdiagonal is not set to zero. It is used to store the vectors  $\mathbf{v}_k$  from which one can generate the reflectors  $Q_k = I - \beta \mathbf{v}_k \mathbf{v}_k^T$  whenever necessary.

---

**Algorithm 3.2** Hessenberg reduction of a matrix, `hessenberg(A)`


---

```

1: for  $k = 1$  to  $n - 2$  do
2:    $x = A(k + 1 : m, k)$ 
3:    $[\mathbf{v}, \beta] = \text{house}(x)$ 
4:    $A(k + 1 : n, k : n) = (I - \beta \mathbf{v}_k \mathbf{v}_k^T) A(k + 1 : n, k : n)$ 
5:    $A(1 : n, k + 1 : n) = A(1 : n, k + 1 : n) (I - \beta \mathbf{v}_k \mathbf{v}_k^T)$ 
6: end for

```

---

### 3.2.3 Operation Count for Hessenberg Reduction.

The traditional way to estimate execution time of an algorithm is to count the flops, or floating point operations [7, p. 58]. Each addition, subtraction, multiplication, division or square root counts as one flop. In applying the Householder reflection  $I - \beta \mathbf{v}_k \mathbf{v}_k^T$  to a vector of length  $\ell$  we have  $\ell$  subtractions,  $\ell$  multiplications for the scalar  $\beta$  and  $\ell$  multiplications and  $\ell - 1$  additions for the dot product. Thus, the orthogonal transformation requires

$$= 4\ell - 1 \approx 4\ell \quad \text{flops,}$$

which is 4 flops per entry operated on. The work of the Householder reduction in this algorithm is dominated by the two updates, multiplication by  $Q$  from the right and by  $Q^T$  from the left of the sub-matrices of  $A$ . In the left multiplication we operate on the last  $n - k$  rows and on these rows the first  $k - 1$  columns are zero. Thus, we only operate on the last  $(n - k + 1)$  elements in each row. So, the total flops count for the left multiplication is

$$\sum_{k=1}^{n-2} 4[(n - k + 1)^2 + 1] = \frac{4}{3}n^3 + 2n^2 + \frac{14}{3}n - 28 \approx \frac{4}{3}n^3 \quad \text{flops.}$$

Since there are no zeros to be ignored, the right multiplication affects more elements than the left multiplication. The total number of entries affected by this operation is  $n(n - k)$  and hence the total flop count for the right multiplication is

$$\sum_{k=1}^{n-2} 4n(n - k) = 2n^3 - 2n^2 - 4n \approx 2n^3 \quad \text{flops.}$$

Therefore, the total flops count for the general Hessenberg form is  $\frac{10}{3}n^3$  [23, p.199].



### 3.2.4 Explicit Computation of the Orthogonal Matrix

In most numerical computations it is not necessary to compute the matrix  $Q$  explicitly, but rather its effect on some vector. For example we may need to produce  $Q\mathbf{x}$  or  $Q^T\mathbf{x}$  for a given vector  $\mathbf{x}$  and so Algorithms 3.4 and 3.3 compute these quantities respectively.

---

**Algorithm 3.3** Computation of  $Q^T\mathbf{x}$ 

---

```
for  $k = 1$  to  $n$  do
     $\mathbf{x}(k:n) = \mathbf{x}(k:n) - \beta \mathbf{v}_k \mathbf{v}_k^T \mathbf{x}(k:n)$ 
end for
```

---

---

**Algorithm 3.4** Computation of  $Q\mathbf{x}$ 

---

```
1: for  $k = n$  downto 1 do
2:    $\mathbf{x}(k:n) = \mathbf{x}(k:n) - \beta \mathbf{v}_k \mathbf{v}_k^T \mathbf{x}(k:n)$ 
3: end for
```

---

In Algorithm 3.2 the orthogonal matrix  $Q$  has not been constructed explicitly, instead the vectors  $\mathbf{v}_k$  are stored and can be used to reconstruct  $Q$  if necessary. The explicit computation of  $Q$  needs additional work, but we need it for our purposes here. The matrix  $Q$  is obtained by taking the product of the Householder matrices as in equation (3.5). Two possible algorithms are discussed in [9, p. 213] for the computation of the Householder product matrix  $Q$ .

The *forward accumulation* is given by

```

1:  $Q = I_n$ 
2: for  $k = 1$  to  $n - 2$  do
3:    $Q = QQ_k$ 
4: end for

```

and the *backward accumulation* by

```

1:  $Q = I_n$ 
2: for  $k = n - 2$  downto 1 do
3:    $Q = Q_k Q$ 
4: end for

```

The leading  $(k - 1) \times (k - 1)$  portion of  $Q_k$  is the identity. Hence, at the beginning of the backward accumulation  $Q$  is “mostly identity” and it gradually becomes a full matrix as the iteration progresses. Therefore, the backward accumulation is cheaper than the forward [9, p. 213]. In this computation, if we apply the general matrix-matrix multiplication to  $Q_k Q$  it will be  $O(n^3)$  operations. Because the computation of  $Q$  needs to be repeated for each Householder reflector  $Q_k$  one will obtain an overall cost of  $O(n^4)$  which is unnecessarily expensive. In the current version of SciPy (version 0.8.0) it is implemented using the Level 3 BLAS (Basic Linear Algebra) GEMM (General Matrix-Matrix Multiplication) which is of  $O(n^4)$  as shown in Figure 3.1.

However, the product  $Q_k Q$  can be written as

$$Q_k Q = Q - \beta \mathbf{v}_k \mathbf{v}_k^T Q,$$

so the main task is to calculate the product  $\beta \mathbf{v}_k \mathbf{v}_k^T Q$ . There are good and bad ways to do this. The first good thing we can do is to absorb the scalar  $\beta$  into one of the vectors. Let  $\mathbf{u}_k = \beta \mathbf{v}_k$ , so that  $Q_k Q = Q - \mathbf{u}_k \mathbf{v}_k^T Q$ . The amount of work required to compute  $\mathbf{u}_k \mathbf{v}_k^T Q$  depends dramatically upon the order in which the operations are performed [24, p. 200]. If we compute  $(\mathbf{u}_k \mathbf{v}_k^T) Q$ , then we get an intermediate  $n \times n$ -matrix and the total computation is  $O(n^3)$  operations. But, if we compute the product  $\mathbf{u}_k (\mathbf{v}_k^T Q)$ , the intermediate result will be a  $1 \times n$ -matrix and the total computation is an  $O(n^2)$  operation. The second way requires much less space and arithmetic than the first arrangement. Therefore, the efficient computation of  $Q_k Q$  is to compute  $Q - \mathbf{u}_k (\mathbf{v}_k^T Q)$  and this can be summarized by Algorithm 3.5, which overwrites  $Q$  with  $Q_k Q$ .

**Algorithm 3.5** Explicit computation of  $Q$ 

- 1:  $\mathbf{u}_k = \beta \mathbf{v}_k$
- 2:  $\mathbf{w}_k = Q_k^T \mathbf{v}_k$
- 3:  $Q = Q - \mathbf{u}_k \mathbf{w}_k^T$

In this algorithm

$$\mathbf{w}_k^T = \mathbf{v}_k^T Q_k,$$

is a matrix-vector multiplication and

$$Q = Q - \mathbf{u}_k \mathbf{w}_k^T.$$

Thus, the Level 2 BLAS functions GEMV (General Matrix-Vector Multiplication) and GER (General Reduction) can be used in the implementation to compute  $\mathbf{v}_k^T Q$  and  $Q - \mathbf{u}_k \mathbf{w}_k^T$  both of which are  $O(n^2)$  operations. Therefore, the overall computation of  $Q$  will be  $O(n^3)$ . More precisely, it requires about  $4(n^2(n-2) - n(n-2)^2 + (n-2)^3/3) \approx \frac{4n^3}{3}$  flops [9, p. 213].

We implemented this approach and compared it with the built-in `hessenberg` function in SciPy. Figure 3.1 below confirms that the built-in `hessenberg` function is of  $O(n^4)$  and our improved function is of  $O(n^3)$ .

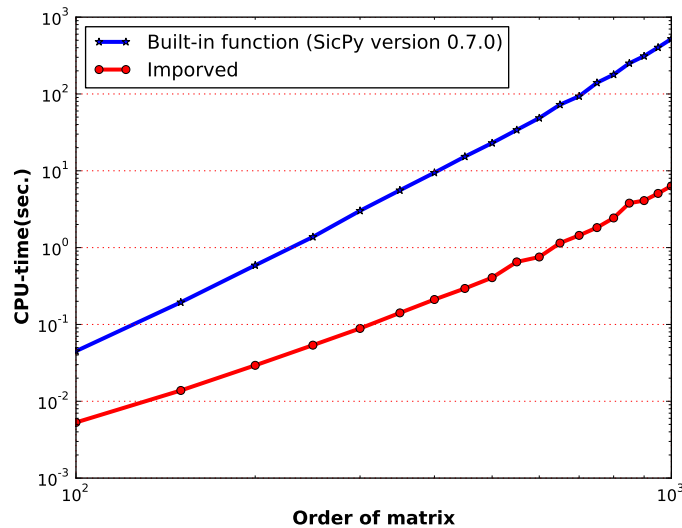


Figure 3.1. *CPU-time vs. order of matrix for the explicit computation of the orthogonal matrix in the Hessenberg reduction.*

The computation of the Hessenberg reduction requires  $\frac{10}{3}n^3$  flops and the explicit computa-

tion of the orthogonal matrix requires an additional  $\frac{4}{3}n^3$  flops. Thus, the total computation requires  $\frac{14}{3}n^3$  flops. As a result, the ratio of the computational time of the Hessenberg reduction with explicit computation and without explicit computation of the orthogonal matrix is approximately 1.4 in theory. As we see from Figure 3.2 this ratio in the built-in function around 150 which is far from the ratio in the theory and this is another confirmation that the built-in function is inefficiently implemented. In addition, this ratio is a linear function of the order of the matrix as the *Ratio of CPU-time* plot in Figure 3.2 shows. However, Figure 3.3 shows that our improved `hessenberg` function gives a ratio around 2.24 which is close enough to the theoretical ratio.

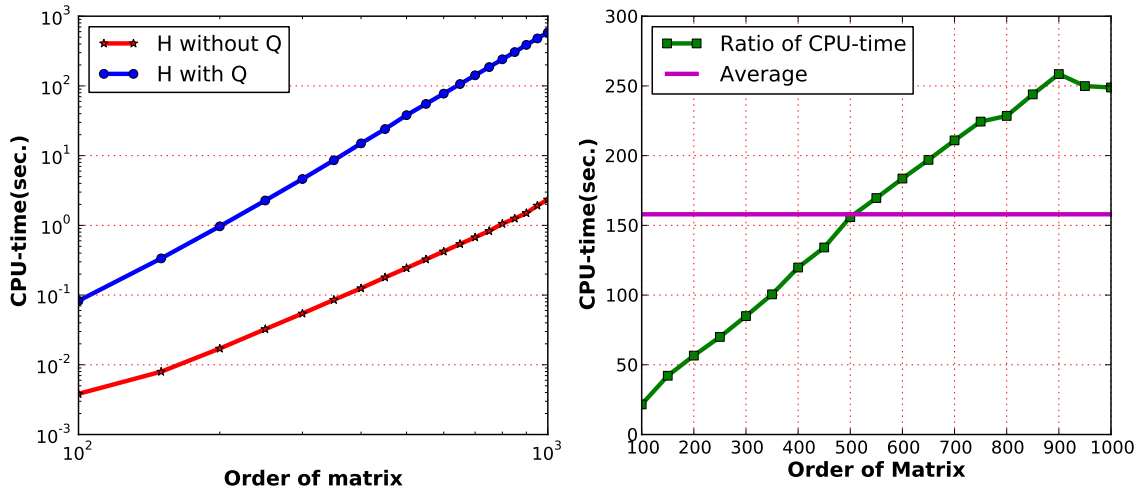


Figure 3.2. Built-in `hessenberg` function: *CPU-time vs. order of matrix of the Hessenberg decomposition with and without the explicit computation of the orthogonal matrix. The ratio of the CPU-time, the green line, is a linear function of the order of the matrix.*

### 3.3 Hessenberg Reduction Method

Let us now return to the main problem of this chapter, namely the solution of shifted linear systems involving dense matrices. That is,

$$(z_k I - A)\mathbf{x} = \mathbf{b}, \quad (3.6)$$

where  $z_k$  is a nodal point on the parabolic contour,  $A \in \mathbb{R}^{n \times n}$  is a dense matrix and  $\mathbf{x}, \mathbf{b} \in \mathbb{R}^N$ .

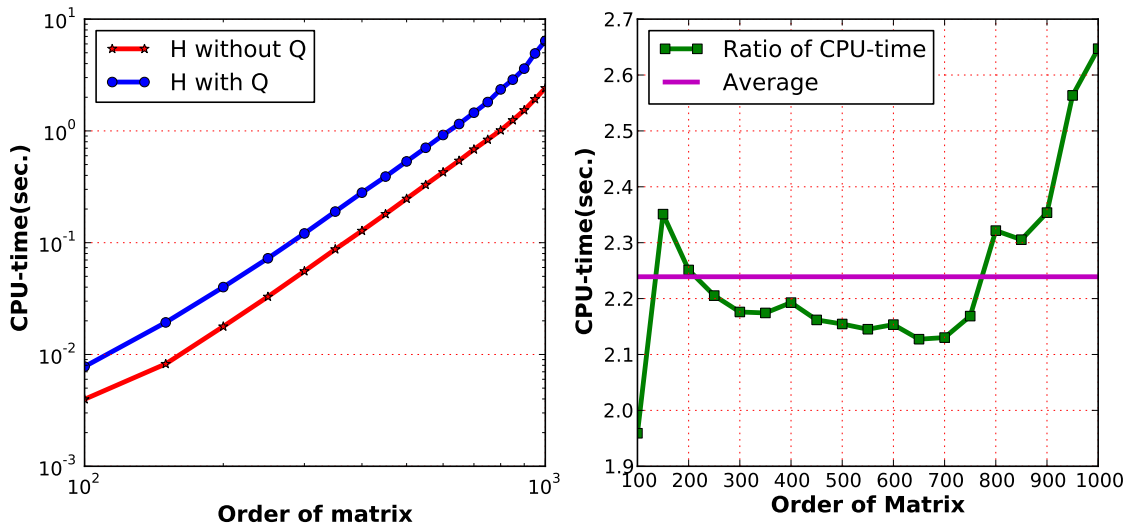


Figure 3.3. Improved hessenberg function: *CPU-time vs. order of matrix of the Hessenberg decomposition with and without the explicit computation of the orthogonal matrix.*

We now show how this system can be solved using the Hessenberg factorization. From (3.4) we get

$$(z_k I - QHQ^T)\mathbf{x} = \mathbf{b},$$

which can be rewritten as

$$(z_k I - H)\mathbf{y} = \mathbf{c}, \quad (3.7)$$

where  $\mathbf{y} = Q^T \mathbf{x}$ , which gives  $\mathbf{x} = Q\mathbf{y}$ , and  $\mathbf{c} = Q^T \mathbf{b}$ . This approach changes the problem of solving a shifted linear system with a dense matrix into solving a shifted linear system with an almost upper-triangular matrix with some additional matrix-vector multiplications. Then sparse solvers can be applied to solve the system (3.7) for each  $z_k$  which is much faster than applying a direct linear system solver to (3.6). We call this approach the Hessenberg Reduction Method (HRM).

We solve (2.2) by the spectral method followed by the contour integral method. The shifted linear systems are solved by the HRM. Figure 3.4 shows the solution of the 1D heat equation (2.2). In the next section we compare the HRM and DLSS numerically.

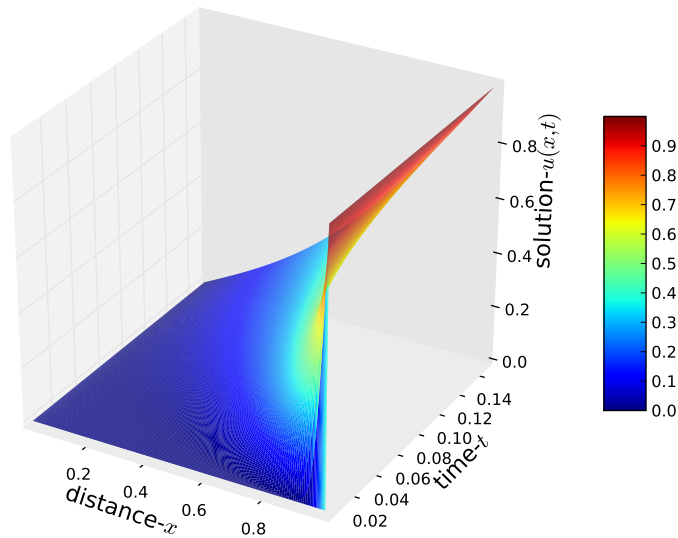


Figure 3.4. *The numerical solution of the 1D model problem (2.2) by taking  $N = 128$  collocation points and  $M = 12$  nodes on the parabolic contour for  $t \in [0.0, 0.15]$ . The shifted linear systems from the contour integral method are solved by the Hessenberg Reduction Method.*

### 3.4 Numerical Comparison of HRM and DLSS

The computational cost of solving the shifted linear systems (3.6) is dominated by the Hessenberg decomposition (3.4). However, only one Hessenberg reduction is required to solve all the shifted linear systems for the solution of the PDE. As a result the overall computational cost will be reduced.

Since both HRM and DLSS are direct methods, they result in the exact solution of the shifted linear systems in exact arithmetic. We compute the solution of equation (2.18) using the built-in `expm` function in SciPy and used these solutions as reference to compare the relative errors in the time-discretization. Thus, the accuracy of formula (2.23) as approximation to the solution of (2.18) depends on the choice of  $M$ , the number of nodes on the contour. To compare the HRM and DLSS we only need to consider the computation time against the number of nodes on the contour. Therefore, we plot the CPU-time vs. number of nodes by taking  $N = 128$ ,  $N = 256$  and  $N = 512$  collocation points in the space discretization. Figure 3.5 shows that the CPU-time of the HRM is less than that of the DLSS for almost all choices of  $M$  and regardless of the order of the matrix. As we see

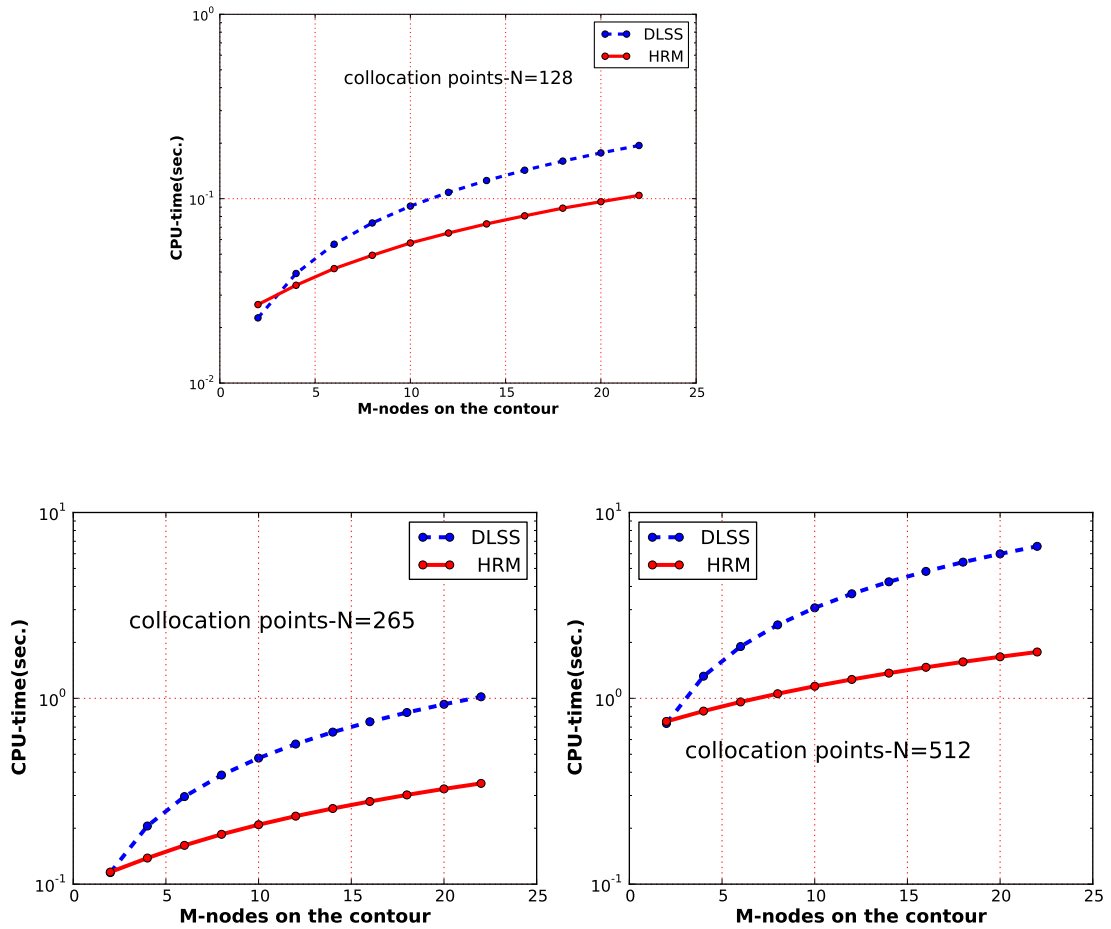


Figure 3.5. *CPU-time vs. number of nodes on the contour for  $N = 128$ ,  $N = 256$  and  $N = 512$  collocation points of the 1D heat equation model problem (2.2) using the direct linear system solver and the Hessenberg reduction method.*

from Figure 3.5 the CPU-time difference is not significant as we increase the order of the matrix. The accuracy of the contour integral method depends on  $M$ , however when  $M$  is large the roundoff error will increase [26] and so a moderately small value of  $M$  is needed for better accuracy.

In the above computations we only considered the time discretization error in the contour approximation. The total error also contains the error in the space discretization. To compute the total error we used the analytic solution (2.3). Here we compare the Hessenberg reduction method with the direct linear system solver by computing the total error in the maximum norm against the number of nodes on the contour. As we see from Figure

3.6 both the direct system solver and the Hessenberg reduction method yield in the same accuracy.

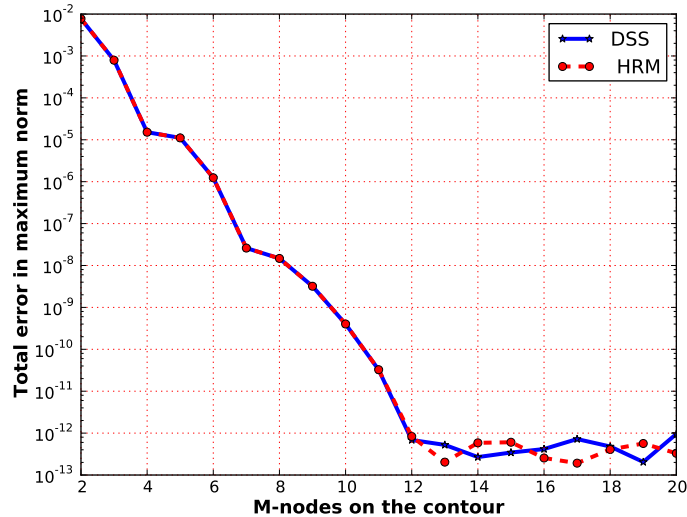


Figure 3.6. Total error in maximum norm vs. number of nodes on the contour for  $N = 256$  collocation points of the 1D heat equation model problem (2.2) using the direct linear system solver and the Hessenberg method.

We also plot the total error in the maximum norm vs. CPU-time for  $2 \leq M \leq 20$ , where  $M$  is the number of nodes on the contour. In Figure 3.7 we can see that the Hessenberg reduction method is more efficient than the built-in direct linear system solver.

## 3.5 Summary and Conclusion

In this chapter we discussed the Hessenberg decomposition of a matrix and used it to solve shifted linear systems of equations. We found that the built-in `hessenberg` function in SciPy is inefficiently implemented. We improved this function and compared it with the built-in function.

Applying spectral methods followed by the contour approximation to the one dimensional heat equation model problem results in shifted linear systems of equations with a dense matrix. Since Gaussian elimination of the shifted matrix is needed for each  $z_k$ , the direct linear system solver is computationally expensive. When the coefficient matrix is dense, we



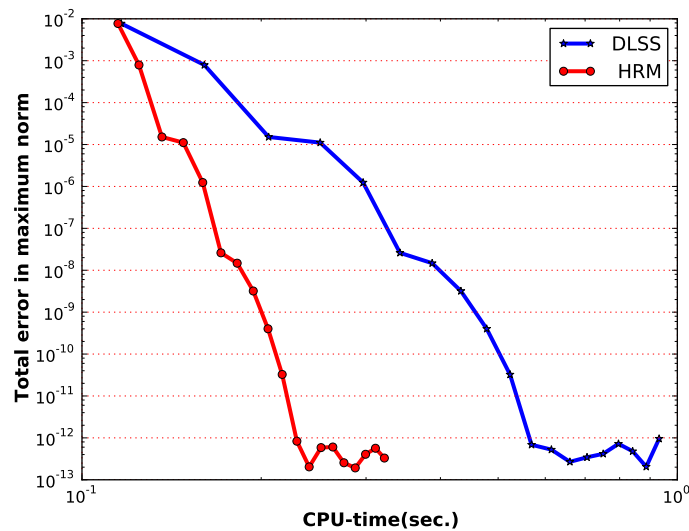


Figure 3.7. Total error in the maximum norm vs. CPU-time(sec.) for  $N = 256$  collocation points of the 1D heat equation model problem (2.2) using the direct linear system solver and the Hessenberg method.

transform the matrix into Hessenberg form and we use sparse solvers to solve the almost triangular shifted systems. The Hessenberg reduction is computationally expensive, but only one reduction is required to solve all systems. As a result the overall computation cost is reduced as compared to the direct solver.

# Chapter 4

## Sparse shifted linear systems

*“The name of the new game is **iteration with preconditioning**.”*  
Lloyd N. Trefethen [23].

### 4.1 Introduction

In Chapter 3 we discussed efficient methods for solving shifted linear systems with dense matrices and in this chapter we discuss sparse matrices. As discussed in Chapter 2, applying finite differences followed by the contour integral method to the two-dimensional heat equation results in shifted linear systems with sparse matrices. In the case when the systems are dense we used the Hessenberg reduction, which yields an almost upper triangular matrix and is solved efficiently using sparse solvers. However, applying Hessenberg reduction to the sparse matrix  $A$  will destroy the sparsity of the matrix, because of the problem of fill-in. Thus, it is not efficient to apply the Hessenberg reduction method to systems with a sparse matrix.

Direct linear system solvers will likewise destroy the pattern of the non-zeros due to fill-in. This needs a large memory and will lead to an unacceptable computing time. For these reasons, large sparse linear systems are commonly solved by iterative methods. These iterative methods are convenient because of three reasons. Firstly, they only require matrix-vector multiplication which is good for sparse matrices. Usually, this operation can be efficiently implemented without waste of memory space. Secondly, the iterative procedure can be stopped part way with a prescribed error tolerance unlike the Householder reflections

in the Hessenberg decomposition. Thirdly, they do not require an explicit representation of the matrix  $A$ , only the action of the matrix on a vector is necessary, i.e., they are matrix-free methods.

In this chapter we discuss efficient methods for solving systems of the form

$$(\alpha I - A)\mathbf{x} = \mathbf{b}, \quad (4.1)$$

where  $\alpha$  is a complex scalar and  $A \in \mathbb{R}^{n \times n}$  is a large sparse matrix. These systems arise in many problems of science and engineering and they typically need to be solved for many values of  $\alpha$ , while  $A$  and  $\mathbf{b}$  remain fixed. As an application we shall solve the two-dimensional heat equation with and without a convection term as defined in Section 2.2. The resulting shifted linear systems will be solved by Krylov subspace methods in this chapter.

Because of their shift-invariance property, Krylov subspace methods are considered as effective methods for solving sparse shifted linear systems of large order. This property allows one to obtain approximate solutions for all values of the parameter by only generating a single approximation space. Thus, the computational time and memory requirements are reduced to a great extent.

This chapter is organized as follows. In Section 4.2 we introduce the projection method called the Krylov subspace method for solving systems of linear equations. We shall briefly discuss how these methods extract an approximate solution from a subspace of  $\mathbb{R}^n$  by projecting the system into a much smaller subspace and imposing orthogonality and minimization constraints. We present the Arnoldi and Lanczos processes, which construct the basis of the Krylov subspace for nonsymmetric and symmetric matrices, respectively. We discuss two methods for solving symmetric problems, the Lanczos Method (LM) and method of Minimal Residuals (MINRES). We also discuss two methods for solving nonsymmetric problems, the Full Orthogonalization Method (FOM) and the method of General Minimal Residuals (GMRES). In Section 4.3 we extend these Krylov methods to solve shifted linear systems. In Section 4.4 we discuss preconditioning and how it improves the convergence of these methods. In Section 4.5 we present numerical results and conclusions.

## 4.2 Krylov Subspace Methods

First, let us discuss general projection methods for a linear system of equations

$$A\mathbf{x} = \mathbf{b}. \quad (4.2)$$

Projection techniques extract an approximate solution for such a system from a subspace of  $\mathbb{R}^n$ . Let  $\mathbb{K}$  be a subspace of  $\mathbb{R}^n$  of dimension  $m$  such that  $m \ll n$ , which contains the approximate solution of (4.2). The subspace  $\mathbb{K}$  is called the search subspace [20, p. 122]. We need to impose  $m$  constraints to extract the approximate solution from the search subspace. These constraints are orthogonality conditions called Petrov-Galerkin conditions [20, p. 124].

Specifically, to extract the approximate solution from  $\mathbb{K}$ , the residual vector  $\mathbf{r} = \mathbf{b} - A\mathbf{x}$  is constrained to be orthogonal to  $m$  linearly independent vectors [20, p. 123]. Let  $\mathbb{L}$  be a subspace of linearly independent vectors, called the subspace of constraints. Thus, general projection methods find an approximation  $\hat{\mathbf{x}} \in \mathbb{K}$  by imposing the condition that the new residual is orthogonal to  $\mathbb{L}$ . These iteration methods require an initial guess to the solution. If  $\mathbf{x}_0$  is used as an initial guess, then the approximate solution must be found in the space  $\mathbf{x}_0 + \mathbb{K}$  instead of the vector space  $\mathbb{K}$ . Thus, we need to find  $\hat{\mathbf{x}} \in \mathbf{x}_0 + \mathbb{K}$ , such that

$$\mathbf{b} - A\hat{\mathbf{x}} \perp \mathbb{L}. \quad (4.3)$$

As iterative methods, these projection techniques use a succession of such projections. Typically, a new pair of subspaces  $\mathbb{K}$  and  $\mathbb{L}$  and initial guess  $\mathbf{x}_0$  equal to the most recent approximation are used. To illustrate the general technique, let us write the basis of  $\mathbb{K}$  in matrix form as  $V_m = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m]$  and the basis of  $\mathbb{L}$  as  $W_m = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_m]$ . If the current approximate solution is

$$\mathbf{x}_m = \mathbf{x}_0 + V_m \mathbf{y}_m, \quad (4.4)$$

then the orthogonality condition (4.3) leads to the following systems of equations for the vector  $\mathbf{y}_m$ ,

$$W_m^T A V_m \mathbf{y}_m = W_m^T \mathbf{r}_0, \quad \mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0, \quad (4.5)$$

which results in

$$\mathbf{y}_m = (W_m^T A V_m)^{-1} W_m^T \mathbf{r}_0,$$

with the assumption that the matrix  $W_m^T A V_m$  is nonsingular. Thus, the expression for the approximate solution is given by

$$\mathbf{x}_m = \mathbf{x}_0 + V_m (W_m^T A V_m)^{-1} W_m^T \mathbf{r}_0.$$

In general, the projection method that approximates the solution of the linear system (4.2) is summarized by the following algorithm.

---

**Algorithm 4.1** General projection method [20, p. 125].

---

- 1: choose  $\mathbf{x}_0 \in \mathbb{R}^n$
  - 2: **for**  $m = 1, 2, \dots$  until convergence **do**
  - 3:   choose subspace  $\mathbb{K}$  with basis  $V_m = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m]$
  - 4:   and subspace  $\mathbb{L}$  with basis  $W_m = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_m]$
  - 5:    $\mathbf{r}_{m-1} = \mathbf{b} - A\mathbf{x}_{m-1}$
  - 6:    $\mathbf{y}_m = (W_m^T A V_m)^{-1} W_m^T \mathbf{r}_{m-1}$
  - 7:    $\mathbf{x}_m = \mathbf{x}_{m-1} + V_m \mathbf{y}_m$
  - 8: **end for**
- 

Depending on the choice of the search subspace  $\mathbb{K}$  and the constraint subspace  $\mathbb{L}$  we get different projection methods. However, all these methods share the property that the exact solution can be obtained after  $k$  iterations, where  $k \leq n$  depends on the coefficient matrix and the initial residual  $\mathbf{r}_0$ , in absence of roundoff errors. If the  $\mathbb{K}$  is the same as  $\mathbb{L}$ , then the projection is called an orthogonal projection method, otherwise it is called an oblique projection method [20, p. 122]. In the next section we introduce a particular choice of search subspace  $\mathbb{K}$  called the Krylov subspace.

### 4.2.1 The Search Subspace

**Definition 4.2.1.** Let  $A \in \mathbb{R}^{n \times n}$  and  $\mathbf{v} \in \mathbb{R}^n$  with  $\mathbf{v} \neq \mathbf{0}$ , then the subspace

$$\mathbb{K}_k(A, \mathbf{v}) = \text{span}\{\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, \dots, A^{k-1}\mathbf{v}\}$$

is called the  $k^{\text{th}}$  Krylov subspace associated with  $A$  and  $\mathbf{v}$ .

We now list a few elementary properties of Krylov subspaces. For the proofs, we refer to [20, p. 144–145].

**Lemma 1.** Let  $A \in \mathbb{R}^{n \times n}$  and  $\mathbf{v} \in \mathbb{R}^n$  with  $\mathbf{v} \neq \mathbf{0}$ . Then,

1. By construction  $\mathbb{K}_k(A, \mathbf{v}) \subseteq \mathbb{K}_{k+1}(A, \mathbf{v})$  and  $A\mathbb{K}_k(A, \mathbf{v}) \subseteq \mathbb{K}_{k+1}(A, \mathbf{v})$ .
2. A Krylov subspace is invariant under scaling, i.e., if  $\sigma \neq 0$ , then  $\mathbb{K}_k(\sigma A, \mathbf{v}) = \mathbb{K}_k(A, \sigma \mathbf{v}) = \mathbb{K}_k(A, \mathbf{v})$ .
3. A Krylov subspace is invariant under shifting, i.e., for any  $c$   $\mathbb{K}_k(A - cI, \mathbf{v}) = \mathbb{K}_k(A, \mathbf{v})$ .

**Theorem 4.2.2.** *The subspace  $\mathbb{K}_k(A, \mathbf{v})$  can be written as*

$$\mathbb{K}_k(A, \mathbf{v}) = \{p(A)\mathbf{v} : p \text{ is a polynomial of degree at most } k - 1\}.$$

**Theorem 4.2.3.** *The Krylov sequence  $\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, \dots$  terminates at  $\eta$  if  $\eta$  is the smallest integer such that*

$$\mathbb{K}_{\eta+i}(A, \mathbf{v}) = \mathbb{K}_\eta(A, \mathbf{v}),$$

and

$$\dim[\mathbb{K}_{\eta+i}(A, \mathbf{v})] = \dim[\mathbb{K}_\eta(A, \mathbf{v})], \quad i \geq 1.$$

A Krylov subspace method is a projection method for which the search subspace is the Krylov subspace. Here, we want to find an approximate solution within the Krylov subspace  $\mathbb{K}_k(A, \mathbf{r}_0)$  and a stable iterative method. In order to have a stable method we need to construct a suitable basis for the subspace. The most obvious approach of constructing the basis of  $\mathbb{K}_k(A, \mathbf{r}_0)$  is repeatedly multiplying the starting vector  $\mathbf{r}_0$  by  $A$  [10, p. 530]. This procedure is numerically unstable since as  $j$  grows large vectors of the form  $A^j\mathbf{r}_0$  approach the dominant eigenvector of  $A$  and become linearly dependent in finite-precision arithmetic [10, p. 530]. Below, we discuss two methods for constructing an orthogonal basis of the Krylov subspace, namely the Arnoldi and Lanczos processes when  $A$  is nonsymmetric and symmetric, respectively.

### Arnoldi Process

The Arnoldi method constructs an orthonormal basis  $V_m = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m]$  of  $\mathbb{K}_m(A, \mathbf{r}_0)$  using the standard Gram-Schmidt orthogonalization process [20, p. 146]. At each step of the algorithm the previous Arnoldi vector  $\mathbf{v}_j$  is multiplied by  $A$  and then the result is orthogonalized against the previous vectors  $[\mathbf{v}_1, \dots, \mathbf{v}_j]$ . However, in the presence of roundoff errors orthogonality will be lost and more reliable algorithm is based on the modified Gram-Schmidt procedure [20, p. 148].

The following properties of Algorithm 4.2 are proved in [20, p. 146–148].

---

**Algorithm 4.2** Arnoldi process-Modified Gram-Schmidt [20, p. 148].

---

```

1: choose a vector  $\mathbf{v}_1$  of norm 1
2: for  $j = 1, 2, 3, \dots, m$  do
3:    $\mathbf{w}_j = A\mathbf{v}_j$ 
4:   for  $i = 1, 2, \dots, j$  do
5:      $h_{ij} = (\mathbf{w}_j, \mathbf{v}_i)$ 
6:      $\mathbf{w}_j = \mathbf{w}_j - h_{ij}\mathbf{v}_i$ 
7:   end for
8:    $h_{j+1,j} = \|\mathbf{w}_j\|_2$ . If  $h_{j+1,j} = 0$ , then stop
9:    $\mathbf{v}_{j+1} = \mathbf{w}_j/h_{j+1,j}$ 
10: end for
    
```

---

**Proposition 4.2.4.** *Assume that Algorithm 4.2 does not stop before the  $m$ -th step. Then the vectors  $[\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m]$  form an orthonormal basis of the Krylov subspace*

$$\mathbb{K}_m = \text{span}\{\mathbf{v}_1, A\mathbf{v}_1, A^2\mathbf{v}_1, \dots, A^m\mathbf{v}_1\}.$$

**Proposition 4.2.5.** *Denote by  $V_m$ , the  $n \times m$  matrix with column vectors  $[\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m]$ , by  $\overline{H}_m$  the  $(m+1) \times m$  Hessenberg matrix whose nonzero entries  $h_{ij}$  are defined by Algorithm 4.2, and by  $H_m$  the matrix obtained from  $\overline{H}_m$  by deleting its last row. Then the following relations hold:*

$$\begin{aligned} AV_m &= V_m H_m + w_m \mathbf{e}_m^T \\ &= V_{m+1} \overline{H}_m, \\ V_m^T AV_m &= H_m. \end{aligned} \tag{4.6}$$

Equation (4.6) is called the Arnoldi relation. The proof of the above propositions is as follows. We start with unit vector  $\mathbf{v}_1$  and at the  $j^{\text{th}}$  step of the Arnoldi process we have

$$\mathbf{w}_j = A\mathbf{v}_j - \sum_{i=1}^j h_{i,j} \mathbf{v}_i,$$

where

$$h_{i,j} = (\mathbf{w}_j, \mathbf{v}_i),$$

and

$$\mathbf{v}_{j+1} = \frac{\mathbf{w}_j}{h_{j+1,j}}, \quad h_{j+1,j} = \|\mathbf{w}_j\|_2.$$

Thus, we have

$$\begin{aligned} A\mathbf{v}_j &= \sum_{i=1}^j h_{i,j}\mathbf{v}_i + h_{j+1,j}\mathbf{v}_{j+1} \\ &= \sum_{i=1}^{j+1} h_{i,j}\mathbf{v}_i. \end{aligned}$$

Since the Arnoldi process runs for  $j = 1, 2, \dots, m$  and writing it in matrix form we get

$$AV_m = V_{m+1}\overline{H}_m.$$

The merit of Arnoldi's method is the orthogonality of the basis. The reason is that orthogonality is a key feature for numerical stability. However, its computation is expensive in terms of both arithmetic and storage requirement. Since all the previous Arnoldi vectors are involved the Arnoldi process is said to be based on long recurrences.

### Lanczos Process

If the matrix  $A$  is symmetric in the Arnoldi process, once  $A\mathbf{v}_j$  is orthogonal to  $\mathbf{v}_j$  and  $\mathbf{v}_{j-1}$  then it is automatically orthogonal to  $\mathbf{v}_i$ , for  $i < j - 1$  and the Arnoldi process simplifies to a short three-term recurrence called the Lanczos process [21]. As a result, the Hessenberg matrix  $H_m$  becomes a symmetric tridiagonal matrix  $T_m$ . The Lanczos process is summarized below in Algorithm 4.3.

---

**Algorithm 4.3** Lanczos process [20, p. 173].

---

- 1: choose a vector  $\mathbf{v}_1$  of norm 1 and set  $\beta_1 \equiv 0, \mathbf{v}_0 \equiv 0$
  - 2: **for**  $j = 1, 2, \dots, m$  **do**
  - 3:    $\mathbf{w}_j = A\mathbf{v}_j - \beta_j\mathbf{v}_{j-1}$
  - 4:    $\alpha_j = (\mathbf{w}_j, \mathbf{v}_j)$
  - 5:    $\mathbf{w}_j = \mathbf{w}_j - \alpha_j\mathbf{v}_j$
  - 6:    $\beta_{j+1} = \|\mathbf{w}_j\|_2$ . If  $\beta_{j+1} = 0$ , then stop
  - 7:    $\mathbf{v}_{j+1} = \mathbf{w}_j/\beta_{j+1}$
  - 8: **end for**
-



In Algorithm 4.3 we collect the  $\{\alpha_j\}$  and  $\{\beta_j\}$  to form the tridiagonal matrix

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & & & \beta_m & \alpha_m \end{pmatrix}.$$

In addition, the Arnoldi relation (4.6) becomes

$$AV_m = V_m T_m + w_m \mathbf{e}_m^T, \quad (4.7)$$

which implies that

$$V_m^T AV_m = T_m,$$

and is called the Lanczos relation.

So far, we have introduced the search subspace  $\mathbb{K}_m$  and methods for constructing the orthonormal basis of  $\mathbb{K}_m$ . Next, we present the subspace of constraints  $\mathbb{L}_m$  for the Krylov subspace method.

## 4.2.2 The Subspace of Constraints

A Krylov subspace method is a projection method that extracts the approximate solution from the Krylov subspace  $\mathbb{K}_m(A, \mathbf{r}_0)$  based on the constraint that the associated residual vector is orthogonal to a subspace  $\mathbb{L}_m$ . Here, we present two well known and commonly used subspaces of constraints. We consider an orthogonal projection method where the constraint subspace  $\mathbb{L}_m$  is the Krylov subspace itself, i.e.,  $\mathbb{L}_m = \mathbb{K}_m$ , as well as the choice  $\mathbb{L}_m = A\mathbb{K}_m$ .

We are now in a position to solve the system of linear equations (4.2) by Krylov subspace methods. In the next two subsections we discuss methods for solving nonsymmetric systems.

### 4.2.3 Full Orthogonalization Method

Given a large nonsymmetric matrix  $A \in \mathbb{R}^{n \times n}$  and an initial approximation  $\mathbf{x}_0$  to the system (4.2), we apply the Krylov subspace method with  $\mathbb{L}_m = \mathbb{K}_m$ . When we apply the Galerkin conditions there is no need to generate the basis  $W_m$  of  $\mathbb{L}_m$  as the two subspaces  $\mathbb{K}_m$  and  $\mathbb{L}_m$  are the same. Thus, we need to find an approximate solution  $\mathbf{x}_m$  from the subspace  $\mathbf{x}_0 + \mathbb{K}_m(A, \mathbf{r}_0)$  by imposing the Galerkin condition that the current residual is orthogonal to  $\mathbb{K}_m$ , i.e.,

$$\mathbf{r}_m \perp \mathbb{K}_m. \quad (4.8)$$

If we let  $\mathbf{v}_1 = \mathbf{r}_0 / \|\mathbf{r}_0\|_2$  in the Arnoldi process and  $\beta = \|\mathbf{r}_0\|_2$ , then from equation (4.6) we get

$$V_m^T A V_m = H_m, \quad (4.9)$$

and

$$V_m^T \mathbf{r}_0 = V_m^T (\beta \mathbf{v}_1) = \beta \mathbf{e}_1, \quad (4.10)$$

where  $\mathbf{e}_1 = (1, 0, 0, \dots, 0)^T$  is the first column of the identity matrix of order  $m$ . Replacing  $W_m^T$  by  $V_m^T$  in equation (4.5) and using (4.9) and (4.10) we get

$$H_m \mathbf{y}_m = \beta \mathbf{e}_1. \quad (4.11)$$

Therefore, the approximate solution of (4.2) in  $\mathbf{x}_0 + \mathbb{K}_m$  is given by

$$\mathbf{x}_m = \mathbf{x}_0 + V_m \mathbf{y}_m, \quad (4.12)$$

where

$$\mathbf{y}_m = H_m^{-1}(\beta \mathbf{e}_1). \quad (4.13)$$

Note that a Hessenberg system of the form (4.11) is to be solved for  $\mathbf{y}_m$  at each iteration of the Arnoldi process. Furthermore, the system (4.11) is much smaller than the original system whenever  $m \ll n$ . Thus, this method saves in storage space and computation time. The method based on this approach is called the Full Orthogonalization Method (FOM) and the practical implementation of the resulting method is given below in Algorithm 4.4.

The important feature of this method is that the residual and its norm are readily available.

---

**Algorithm 4.4** Full Orthogonalization Method (FOM) [20, p. 152]
 

---

- 1: choose  $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ ,  $\beta := \|\mathbf{r}_0\|_2$ , and  $\mathbf{v}_1 := \mathbf{r}_0/\beta$
  - 2: define the  $(m+1) \times m$  matrix  $\overline{H}_m = \{h\}$  set  $\overline{H}_m := 0$
  - 3: **for**  $j = 1, 2, 3, \dots, m$  **do**
  - 4:  $\mathbf{w}_j = A\mathbf{v}_j$
  - 5: **for**  $i = 1, 2, \dots, j$  **do**
  - 6:  $h_{ij} = (\mathbf{w}_j, \mathbf{v}_i)$
  - 7:  $\mathbf{w}_j = \mathbf{w}_j - h_{ij}\mathbf{v}_i$
  - 8: **end for**
  - 9:  $h_{j+1,j} = \|\mathbf{w}_j\|_2$ . If  $h_{j+1,j} = 0$ , set  $j = m$  and Goto 12
  - 10:  $\mathbf{v}_{j+1} = \mathbf{w}_j/h_{j+1,j}$
  - 11: **end for**
  - 12: solve  $H_m\mathbf{y}_m = \beta\mathbf{e}_1$
  - 13:  $\mathbf{x}_m = \mathbf{x}_0 + V_m\mathbf{y}_m$ .
- 

Using the Arnoldi relation (4.6) we can write

$$\mathbf{r}_m = \mathbf{b} - AV_m\mathbf{y}_m = \beta\mathbf{v}_1 - V_m H_m \mathbf{y}_m - h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^T \mathbf{y}_m.$$

Using equation (4.13) we get

$$\mathbf{r}_m = -h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^T \mathbf{y}_m,$$

and hence, the norm of the residual is given as

$$\|\mathbf{r}_m\|_2 = h_{m+1,m}|\mathbf{e}_m^T \mathbf{y}_m|. \quad (4.14)$$

Next, we discuss another method of solving nonsymmetric linear systems based on minimization of the residual called the General Minimal Residual method.

#### 4.2.4 General Minimal Residual Method

The General Minimal RESidual Method (GMRES) is used to solve nonsymmetric linear systems [20, p. 157]. In GMRES the constraint to approximate the solution of (4.2) is the minimization of the residual over all the possible vectors in the Krylov subspace [1, 21]. That is, we want to find

$$\mathbf{x}_m = \mathbf{x}_0 + V_m\mathbf{y}_m,$$

for some  $\mathbf{y}_m \in \mathbb{R}^m$  such that

$$\|\mathbf{r}_m\|_2 = \|\mathbf{b} - A\mathbf{x}_m\|_2 = \min_{\mathbf{x} \in \mathbf{x}_0 + \mathbb{K}_m} \|\mathbf{b} - A\mathbf{x}\|_2. \quad (4.15)$$

The key to GMRES is the implementation of the solution of the least squares problem (4.15) using an orthonormal basis of the Krylov subspace produced by the Arnoldi procedure. Using the Arnoldi relation (4.6) we have

$$\begin{aligned} \mathbf{r}_m &= \mathbf{b} - A\mathbf{x}_m = \mathbf{b} - A(\mathbf{x}_0 + V_m\mathbf{y}_m) \\ &= \beta\mathbf{v}_1 - V_{m+1}H_{m+1,m}\mathbf{y}_m \\ &= V_{m+1}(\beta\mathbf{e}_1 - H_{m+1,m}\mathbf{y}_m), \end{aligned}$$

where  $\mathbf{e}_1$  is the first column of the identity matrix of order  $m + 1$ . Since  $V_{m+1}$  has orthonormal columns, the least squares problem (4.15) can be rewritten as

$$\|\mathbf{r}_m\|_2 = \min_{\mathbf{y}_m \in \mathbb{R}^m} \|\beta\mathbf{e}_1 - H_{m+1,m}\mathbf{y}_m\|_2. \quad (4.16)$$

The sequence of norms of the residuals produced by GMRES is non-increasing on the nested Krylov subspaces [21].

Another implementation feature of GMRES is the use of the  $QR$  factorization of the Hessenberg matrix

$$H_{m+1,m} = Q_{m+1,m}R_m, \quad (4.17)$$

where the  $m \times m$  matrix  $R_m$  is upper triangular and  $Q_{m+1,m}$  has orthonormal columns in the least squares problem (4.16). Thus, the least squares problem (4.16) can be written as

$$\|\mathbf{r}_m\|_2 = \min_{\mathbf{y}_m \in \mathbb{R}^m} \|Q_{m+1,m}^T(\beta\mathbf{e}_1) - R_m\mathbf{y}_m\|_2. \quad (4.18)$$

As a result we have

$$R_m\mathbf{y}_m = \beta Q_{m+1,m}^T\mathbf{e}_1. \quad (4.19)$$

In summary, the resulting algorithm is similar to the FOM Algorithm 4.4, the only difference is that the vector  $\mathbf{y}_m$  is the minimizer of the residual. Thus, we have the following algorithm for GMRES.

In the Arnoldi process which is part of FOM and GMRES we need  $m$  small relative to  $n$  and as  $m$  increases the computational cost increases at least as  $O(m^2n)$  because of the Gram-Schmidt orthogonalization [20, p. 153]. The memory cost also increases as  $O(mn)$ .

**Algorithm 4.5** GMRES [20, p. 158]

---

```

1: choose  $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ ,  $\beta := \|\mathbf{r}_0\|_2$ , and  $\mathbf{v}_1 := \mathbf{r}_0/\beta$ 
2: define the  $(m+1) \times m$  matrix  $\overline{H}_m$  and set  $\overline{H}_m := 0$ 
3: for  $j = 1, 2, 3, \dots, m$  do
4:    $\mathbf{w}_j = A\mathbf{v}_j$ 
5:   for  $i = 1, 2, \dots, j$  do
6:      $h_{ij} = (\mathbf{w}_j, \mathbf{v}_i)$ 
7:      $\mathbf{w}_j = \mathbf{w}_j - h_{ij}\mathbf{v}_i$ 
8:   end for
9:    $h_{j+1,j} = \|\mathbf{w}_j\|_2$ . If  $h_{j+1,j} = 0$ , set  $j = m$  and Goto 12
10:   $\mathbf{v}_{j+1} = \mathbf{w}_j/h_{j+1,j}$ 
11: end for
12: compute the  $QR$  factorization of  $H_{m+1}$  as  $H_{m+1,m} = Q_{m+1,m}R_m$ .
13: solve  $R_m\mathbf{y}_m = \beta Q_{m+1,m}^T \mathbf{e}_1$ 
14:  $\mathbf{x}_m = \mathbf{x}_0 + V_m\mathbf{y}_m$ .

```

---

In addition, the orthogonality relations are increasingly contaminated by roundoff error as  $m$  increases. There are methods that alleviate this problem, e.g., restarted or truncated FOM or GMRES, which we do not discuss in this thesis.

In the next subsections we discuss methods for solving linear systems with a symmetric coefficient matrix called the Lanczos Method and the Minimal Residual Method.

### 4.2.5 Lanczos Method

When the Galerkin conditions are applied to symmetric matrices the same formulas as for FOM are obtained and we call it the Lanczos Method. The matrix must be symmetric but it can be definite or indefinite. If the matrix is symmetric positive definite then the Lanczos method is equivalent to the Conjugate Gradient Method (CG), which we do not discuss in this thesis.

Let  $\mathbf{x}_0$  be an initial guess to the solution of (4.2) and  $[\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m]$  be the Lanczos vectors corresponding to the tridiagonal matrix  $T_m$ , then the approximate solution  $\mathbf{x}_m$  from the Krylov subspace  $\mathbb{K}_m$  is given by

$$\mathbf{x}_m = \mathbf{x}_0 + V_m\mathbf{y}_m,$$

where

$$\mathbf{y}_m = T_m^{-1}(\beta \mathbf{e}_1).$$

The algorithm of the Lanczos method is given below.

---

**Algorithm 4.6** Lanczos Method [20, p. 175]

---

- 1: choose  $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ ,  $\beta := \|\mathbf{r}_0\|_2$ , and  $\mathbf{v}_1 := \mathbf{r}_0/\beta$
  - 2: define the  $(m+1) \times m$  matrix  $\overline{T}_m$
  - 3: **for**  $j = 1, 2, 3, \dots, m$  **do**
  - 4:    $\mathbf{w}_j = A\mathbf{v}_j - \beta_j \mathbf{v}_{j-1}$
  - 5:    $\alpha_j = (\mathbf{w}_j, \mathbf{v}_j)$
  - 6:    $\mathbf{w}_j = \mathbf{w}_j - \alpha_j \mathbf{v}_j$
  - 7:    $\beta_{j+1} = \|\mathbf{w}_j\|_2$ . If  $\beta_{j+1} = 0$ , then stop
  - 8:    $\mathbf{v}_{j+1} = \mathbf{w}_j/\beta_{j+1}$
  - 9: **end for**
  - 10: set  $T_m = \text{tridiag}(\beta_i, \alpha_i, \beta_{i+1})$ , and  $V_m = [\mathbf{v}_1, \dots, \mathbf{v}_m]$ .
  - 11: solve  $T_m \mathbf{y}_m = \beta \mathbf{e}_1$
  - 12:  $\mathbf{x}_m = \mathbf{x}_0 + V_m \mathbf{y}_m$ .
- 

In FOM and LM the constraint to solve the linear systems was orthogonality, in the next subsection we discuss a method for solving symmetric linear systems based on a minimum-residual constraint, i.e., the residual  $\|\mathbf{r}_m\|_2$  is minimized.

## 4.2.6 Minimal Residual Method

When the matrix is symmetric and the Lanczos process is used to construct the basis of  $\mathbb{K}_m$  instead of the Arnoldi process, GMRES simplifies to the method called Minimal Residual (MINRES) [16]. Therefore, with MINRES, the approximate solution of (4.2) is found as

$$\mathbf{x}_m = \mathbf{x}_0 + V_m \mathbf{y}_m, \quad (4.20)$$

where  $\mathbf{y}_m$  minimizes  $\|\mathbf{r}_m\|_2$ , i.e.,

$$\|\mathbf{r}_m\|_2 = \min_{\mathbf{y}_m \in \mathbb{R}^m} \|\beta \mathbf{e}_1 - T_{m+1,m} \mathbf{y}_m\|_2. \quad (4.21)$$

The minimization problem (4.21) can be solved by the  $QR$  factorization of the tridiagonal matrix

$$T_{m+1,m} = \tilde{Q}_{m+1,m} \tilde{R}_m, \quad (4.22)$$

as in GMRES and hence we have

$$\|\mathbf{r}_m\|_2 = \min_{\mathbf{y}_m \in \mathbb{R}^m} \|\tilde{Q}_{m+1,m}^T (\beta \mathbf{e}_1) - \tilde{R}_m \mathbf{y}_m\|_2. \quad (4.23)$$

As a result we get

$$\tilde{R}_m \mathbf{y}_m = \beta \tilde{Q}_{m+1,m}^T \mathbf{e}_1. \quad (4.24)$$

Therefore, the approximate solution (4.20) is obtained by solving an upper-triangular system (4.24). The algorithm for MINRES is the same as for GMRES, the only difference is that the Lanczos process is used instead of Arnoldi and it is given below.

---

**Algorithm 4.7** MINRES
 

---

- 1: choose  $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ ,  $\beta := \|\mathbf{r}_0\|_2$ , and  $\mathbf{v}_1 := \mathbf{r}_0/\beta$
  - 2: define the  $(m+1) \times m$  matrix  $\overline{H}_m$  and set  $\overline{H}_m := 0$
  - 3: **for**  $j = 1, 2, 3, \dots, m$  **do**
  - 4:  $\mathbf{w}_j = A\mathbf{v}_j - \beta_j \mathbf{v}_{j-1}$
  - 5:  $\alpha_j = (\mathbf{w}_j, \mathbf{v}_j)$
  - 6:  $\mathbf{w}_j = \mathbf{w}_j - \alpha_j \mathbf{v}_j$
  - 7:  $\beta_{j+1} = \|\mathbf{w}_j\|_2$ . If  $\beta_{j+1} = 0$ , then stop
  - 8:  $\mathbf{v}_{j+1} = \mathbf{w}_j/\beta_{j+1}$
  - 9: **end for**
  - 10: set  $T_m = \text{tridag}(\beta_i, \alpha_i, \beta_{i+1})$ , and  $V_m = [\mathbf{v}_1, \dots, \mathbf{v}_m]$ .
  - 11: compute the  $QR$  factorization of  $T_{m+1}$  as  $T_{m+1,m} = \tilde{Q}_{m+1,m} \tilde{R}_m$ .
  - 12: solve  $\tilde{R}_m \mathbf{y}_m = \beta \tilde{Q}_{m+1,m}^T \mathbf{e}_1$  and compute  $\mathbf{x}_m = \mathbf{x}_0 + V_m \mathbf{y}_m$ .
- 

In this section we have discussed Krylov subspace methods for symmetric and nonsymmetric system of linear equations. In the next section we extend these methods to shifted linear systems.

### 4.3 Shifted Linear Systems

Consider the shifted linear system (4.1). We have seen in Lemma 1 that the Krylov subspaces are invariant under shifting, i.e.,  $\mathbb{K}_m(\alpha I - A, \mathbf{r}_0) = \mathbb{K}_m(A, \mathbf{r}_0)$ . Thus, we construct the Krylov subspace only once and we use it for all parameters  $\alpha$ . In addition, the Arnoldi

relation (4.6) can be written as

$$\begin{aligned} (\alpha I - A)V_m &= \alpha V_m - AV_m = \alpha V_m - (V_m H_m + \mathbf{w}_m \mathbf{e}_m^T) \\ &= V_m(\alpha I_m - H_m) - \mathbf{w}_m \mathbf{e}_m^T, \end{aligned} \quad (4.25)$$

where  $I_m$  is identity matrix of order  $m$ . Thus, in the Arnoldi relation the Hessenberg matrix is shifted,  $\alpha I_m - H_m$ , and the matrices  $V_m$  and  $H_m$  are the same as in (4.6) which are independent of the parameter  $\alpha$ . Similarly, the Lanczos relation (4.7) becomes

$$(\alpha I - A)V_m = V_m(\alpha I_m - T_m) - \mathbf{w}_m \mathbf{e}_m^T. \quad (4.26)$$

Therefore, when we apply the methods discussed above to the shifted system (4.1), the only difference is the computation of  $\mathbf{y}_m$ . Let  $\mathbf{x}_0$  be an initial guess to (4.1),  $\mathbf{v}_1 = \mathbf{r}_0 / \|\mathbf{r}_0\|_2$  be an initial vector for the Arnoldi and Lanczos process and  $\beta = \|\mathbf{r}_0\|_2$ , then the methods based on Arnoldi and Lanczos for the shifted system can be generalized as follows.

### 4.3.1 FOM for shifted linear systems

The approximate solution of the shifted system (4.1) is given by

$$\mathbf{x}_m = \mathbf{x}_0 + V_m \mathbf{y}_m, \quad (4.27)$$

where  $\mathbf{y}_m$  is computed from

$$(\alpha I_m - H_m) \mathbf{y}_m = \beta \mathbf{e}_1.$$

Here  $I_m$  is identity matrix of order  $m$  and  $H_m$  is the Hessenberg matrix from the Arnoldi process. In addition, the residual is given by

$$\mathbf{r}_m = -h_{m+1,m} \mathbf{v}_{m+1} \mathbf{e}_m^T \mathbf{y}_m,$$

and

$$\|\mathbf{r}_m\|_2 = h_{m+1,m} |\mathbf{e}_m^T \mathbf{y}_m|.$$



### 4.3.2 GMRES for shifted linear systems

When we apply GMRES to the shifted linear system only one Krylov subspace is needed as before. The approximate solution to the shifted linear system (4.1) is given by (4.27) where  $\mathbf{y}_m$  is the minimizer of the residual,

$$\|\mathbf{r}_m\|_2 = \min_{\mathbf{y}_m \in \mathbb{R}^m} \|\beta \mathbf{e}_1 - (\alpha \tilde{I} - H_{m+1,m}) \mathbf{y}_m\|_2, \quad (4.28)$$

where

$$\tilde{I} = \begin{bmatrix} I_m \\ \mathbf{0}^T \end{bmatrix}. \quad (4.29)$$

Then, using (4.17) we can write (4.28) as

$$\|\mathbf{r}_m\|_2 = \min_{\mathbf{y}_m \in \mathbb{R}^m} \|\beta Q_{m+1,m}^T \mathbf{e}_1 - (\alpha Q_{m+1,m}^T \tilde{I} - R_m) \mathbf{y}_m\|_2. \quad (4.30)$$

Here, we need only one  $QR$  factorization for each of the parameters  $\alpha$  and the least squares problems (4.30) yields

$$(\alpha Q'_m - R_m) \mathbf{y}_m = \beta Q_{m+1,m} \mathbf{e}_1,$$

where

$$Q'_m = Q_{m+1,m}^T \tilde{I}. \quad (4.31)$$

### 4.3.3 Lanczos method for shifted linear systems

The solution of (4.1) is given by (4.27) where  $\mathbf{y}_m$  is computed from

$$(\alpha I_m - T_m) \mathbf{y}_m = \beta \mathbf{e}_1, \quad (4.32)$$

where  $T_m$  is the tridiagonal matrix from the Lanczos process.

### 4.3.4 MINRES for shifted linear systems

The same formulation follows from GMRES when we apply MINRES to the shifted linear systems (4.1). The solution is given by (4.27) where  $\mathbf{y}_m$  is the minimizer of the residual

$$\|\mathbf{r}_m\|_2 = \min_{\mathbf{y}_m \in \mathbb{R}^m} \|\beta \mathbf{e}_1 - (\alpha \tilde{I} - T_{m+1,m}) \mathbf{y}_m\|_2, \quad (4.33)$$

where  $\tilde{I}$  is defined as in (4.29). The least squares problem (4.33) is solved by computing the  $QR$  factorization of  $T_{m+1,m}$  as given in (4.22). Hence, we have

$$(\alpha\tilde{Q}'_m - R_m)\mathbf{y}_m = \beta\tilde{Q}_{m+1,m}\mathbf{e}_1,$$

where

$$\tilde{Q}'_m = \tilde{Q}_{m+1,m}^T \tilde{I}.$$

In all the above methods the computationally expensive step of generating the basis of the Krylov subspace is done only once for all parameters. In both GMRES and MINRES there is the additional  $QR$  factorization required to solve the least squares problems but this can be done once for all values of the parameters. In practice there is very little difference in computational time of GMRES vs. FOM, and likewise MINRES vs. LM. This can be seen in Figure 4.6 and Figure 4.9. In addition, the memory requirement is independent of the parameters, it only depends on  $m$  and hence is a great computational savings as compared to solving each shifted system separately.

However, since the spectrum of  $\alpha I - A$  spreads over a large region in the complex plane, the Krylov subspace methods converge slowly in general [13, 15, 16]. Let us take a particular  $\alpha$  for the benchmark problem (2.4) and see how the eigenvalues of  $\alpha I - A$  are spread. We plot in Figure 4.1 the eigenvalues  $\alpha I - A$  in the complex plane for  $\alpha = z_2$ , a point on the parabolic contour, of the two dimensional heat equation (2.4).

We can see from Figure 4.1 that the eigenvalues of  $z_2 I - A$  are spread over a large region in the complex plane and hence the Krylov methods converge slowly. Thus, to improve the convergence we need a preconditioner for the shifted systems.

In the next section we discuss general preconditioning of Krylov methods and discuss the shift-invert preconditioning for our benchmark problems.

## 4.4 Preconditioning

As discussed in the previous section, Krylov methods can suffer from slow convergence. To overcome this problem researchers have been trying to find new and better Krylov subspace methods which converge fast with low computational cost. Among the techniques

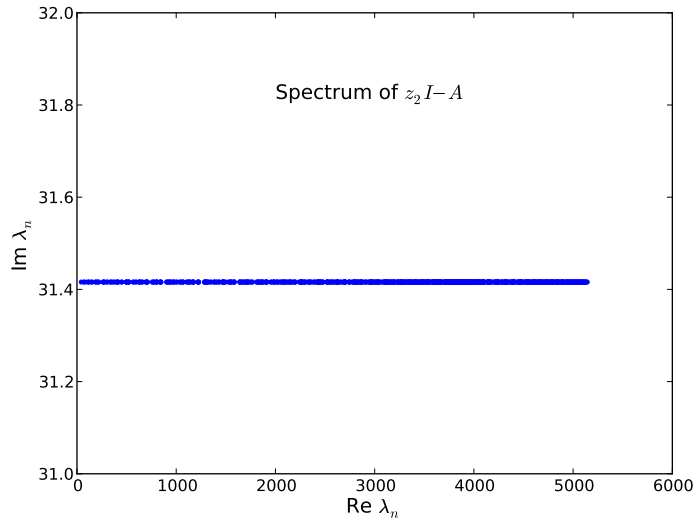


Figure 4.1. *Spectrum of  $z_2 I - A$  for the symmetric model problem of order  $900 \times 900$*

used to get rapid convergence, preconditioning is the most important. Recent research is dedicated to finding better preconditioners rather than trying to improve the Krylov subspace methods. For a complete survey of preconditioners for iterative methods we refer to [1, 2, 23].

Let  $M$  be a preconditioner for the shifted linear system (4.1), then the idea is to solve

$$M^{-1}(\alpha I - A)\mathbf{x} = M^{-1}\mathbf{b}. \quad (4.34)$$

When we solve (4.34) iteratively its convergence depends on the properties of  $M^{-1}(\alpha I - A)$  instead of the properties of  $\alpha I - A$ . If we can construct a good preconditioner and can apply it to all the shifted linear systems, then (4.34) may be solved more rapidly than (4.1). In order to have fast convergence we need a preconditioner that satisfies the following requirements:

1. The eigenvalues of  $M^{-1}(\alpha I - A)$  should be clustered in a small region.
2.  $M^{-1}\mathbf{b}$  should be computed efficiently.

However, the two requirements are opposing one another. If we take  $M = \alpha I - A$ , then the preconditioned system will result in a single eigenvalue of 1, but the computation of

$M^{-1}\mathbf{b}$  is as difficult as the original problem. Also, if we take  $M = I$  then the solution of  $M^{-1}\mathbf{b}$  is fast but the spectrum of  $M^{-1}(\alpha I - A)$  is the same as that of the original problem. Therefore, between these two extremes we need to find a matrix  $M$  that satisfies the two requirements to some extent.

Shift-invert preconditioning is used in [11, 16]. We also use the shift-invert preconditioner

$$M = \sigma I - A, \quad (4.35)$$

for our application as discussed in detail in [16]. Thus, the preconditioned system (4.34) is given by

$$(\sigma I - A)^{-1}(\alpha I - A)\mathbf{x} = (\sigma I - A)^{-1}\mathbf{b}. \quad (4.36)$$

We can rewrite

$$A(\alpha) = (\sigma I - A)^{-1}(\alpha I - A), \quad (4.37)$$

as

$$I + (\alpha - \sigma)(\sigma I - A)^{-1}. \quad (4.38)$$

Let  $B = (\sigma I - A)^{-1}$  and  $\mathbf{d} = (\sigma I - A)^{-1}\mathbf{b}$ , then (4.36) can be written as

$$(I + (\alpha - \sigma)B)\mathbf{x} = \mathbf{d}. \quad (4.39)$$

Since Krylov subspaces are invariant under shifting and scaling the Krylov subspace for (4.39) is the same for all  $\alpha \neq \sigma$  as for  $B$ .

Before we apply the Krylov methods discussed in Section 4.3 to (4.39), let us compute the spectrum of the preconditioner  $(\sigma I - A)$  corresponding to the two dimensional heat equation (2.4). We take  $\sigma = z_0$  and  $\alpha = z_2$  on the contour of the benchmark problem (2.4) and we plot the eigenvalues of  $A(z_2)$  in the complex plane as shown in Figure 4.2. Thus, the eigenvalues of  $A(z_2)$  are clustered near 1. Therefore, the spectrum of  $A(z_2)$  is more favourable for fast convergence than the spectrum of  $z_2 I - A$  shown in Figure 4.1, provided  $\sigma$  is well chosen.

As suggested in [11], it is convenient to use a real matrix preconditioner for slightly more efficiency than the one with complex diagonal. The authors in [11] used  $z_0 I - A$  as a common preconditioner for each systems (2.24). In the practical computation of

$$\mathbf{d} = (z_0 I - A)^{-1} \mathbf{b}, \quad (4.40)$$

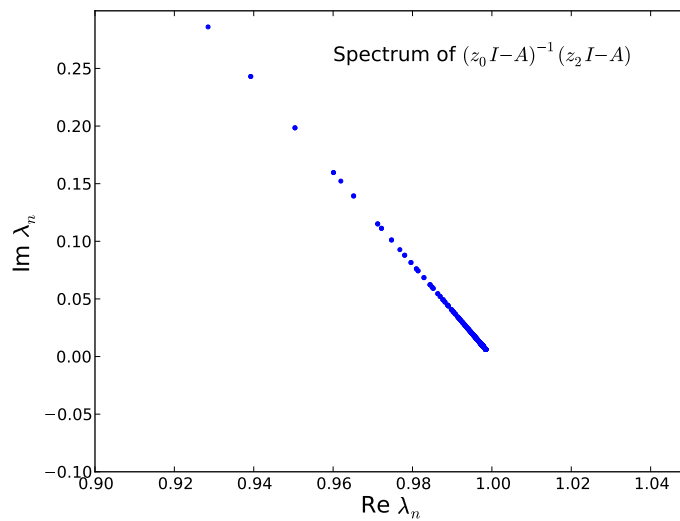


Figure 4.2. Spectrum of the preconditioned matrix  $A(z_2)$  for the symmetric model problem (2.4) of order  $900 \times 900$

we shall use a direct sparse solver. Since we solve (4.40) only once it is not efficient to apply iterative methods here. Therefore,  $(z_0 I - A)^{-1} \mathbf{b}$  is computed efficiently and hence the preconditioner satisfies the second requirement too.

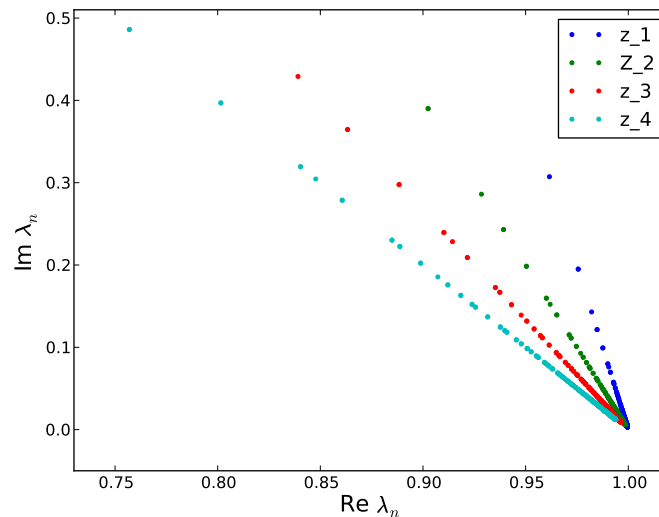


Figure 4.3. Spectrum of  $A(z_k)$ ,  $k = 1, 2, 3, 4$ , for the symmetric problem (2.4) of order  $900 \times 900$

The matrix  $z_0I - A$  is a good preconditioner when it is close to  $z_kI - A$  so that  $(z_0I - A)^{-1}(z_kI - A) \approx I$  and this occurs when  $z_0 \approx z_k$ . Therefore, the matrix  $z_0I - A$  is a good preconditioner for  $z_kI - A$  when  $k = 1, 2$  or  $3$ , but it may not be the case for larger values of  $k$ . In Figure 4.3 we observe that the clustering of the eigenvalues for the preconditioned system is good for small  $k$  but the eigenvalues are spread over large area for large  $k$ . However, due to the decaying factor  $e^{z_k t}$  in (2.8) it is not necessary to compute the systems corresponding to large  $k$  to the same absolute accuracy as the systems corresponding to small  $k$  [11].

We can now apply Krylov methods to the preconditioned systems (4.39). It is required only to compute one Krylov subspace of  $B$  and use this subspace to approximate the solution of (4.39) for each  $\alpha$ . After applying  $m$  steps of the Arnoldi procedure to the matrix  $B$  with  $\mathbf{d}$  as initial vector, the Arnoldi relation

$$V_m^T B V_m = H_m, \quad (4.41)$$

is obtained. The orthogonality condition of the Krylov subspace for (4.39) is simplified to

$$(I + (\alpha - \sigma)H_m)\mathbf{y}_m = \gamma\mathbf{e}_1, \quad (4.42)$$

where  $\gamma = \|\mathbf{d}\|_2$  and the minimization condition is given by

$$\|\mathbf{r}_m\|_2 = \min_{\mathbf{y}_m \in \mathbb{R}^m} \|\beta\mathbf{e}_1 - (I + (\alpha - \sigma)H_{m+1,m})\mathbf{y}_m\|_2. \quad (4.43)$$

When we apply the FOM to (4.39), for example the approximate solution is given by

$$\mathbf{x}_m = \mathbf{x}_0 + V_m\mathbf{y}_m, \quad (4.44)$$

where  $\mathbf{y}_m$  is computed from (4.42). When  $m \ll n$ , the  $m \times m$  Hessenberg-system (4.42) is solved cheaply for each  $\alpha$ .

In addition, at each iteration of the Arnoldi procedure we need to compute a matrix-vector product  $B\mathbf{v}_j$  for  $j = 1, \dots, m$ . Since  $B = (\sigma I - A)^{-1}$ , instead of computing the matrix-vector product by computing an inverse, we need to solve  $(\sigma I - A)\mathbf{w}_j = \mathbf{v}_j$  for each  $\mathbf{v}_j$ . This is done with a prior  $LU$  factorization of  $\sigma I - A$ . If we let  $\sigma I - A = LU$ , then we solve  $(\sigma I - A)\mathbf{w}_j = \mathbf{v}_j$  as

$$L\mathbf{z}_j = \mathbf{v}_j, \quad U\mathbf{w}_j = \mathbf{z}_j, \quad (4.45)$$

for each  $\mathbf{v}_j$ . In the practical implementation we use the function `factorized` in UMFPAK which results in an  $LU$  factorization of a sparse matrix.

## 4.5 Numerical Results

In this section, we discuss numerical comparisons of the built-in Direct Sparse Solver (DSS) in SciPy with the Krylov subspace methods for solving shifted linear systems.

Here, we do not consider the total error as it is dominated by the space discretization error since we used finite differences which is of low order unlike the Chebyshev collocation method. In addition, since we are interested here in solving shifted linear systems we focus only on the time-discretization error.

We therefore compute the solution of equation (2.12) using the built-in `expm` function in SciPy and used these solutions as reference to compare the relative errors in the time-discretization. In all the numerical comparisons the relative error is computed in the maximum norm. A simple and convenient choice  $m = M + 1$  for the number of iterations in the Arnoldi and Lanczos processes is suggested in [11], where  $M$  is the number of nodes on the contour. We take this choice as a reference and also experiment with other choices. In all our numerical experiments we use the 9-point formula in the space discretization and number of spatial grid points  $N = 102$  and hence shifted linear systems of order  $10000 \times 10000$  are solved. We present all results at a fixed time  $t = 0.1$  and  $\kappa = 1$ .

We compare the Krylov methods, Lanczos Method (LM) and MINRES with the DSS, for solving the symmetric systems that arise from the two-dimensional heat equation (2.4). We also compare FOM and GMRES with DSS for solving the nonsymmetric systems from the two-dimensional heat equation with convection term (2.5).

### 4.5.1 Symmetric Problem

We compare the relative error in the time-discretization versus number of nodes on the contour for solving the symmetric model problem (2.4) for different number of iterations in the Lanczos process. Here, we solve the shifted systems (2.24) by LM, MINRES and DSS. The error in solving these systems is the difference between the solution obtained from the

built-in `expm` function in SciPy and the solution from the contour integral method.

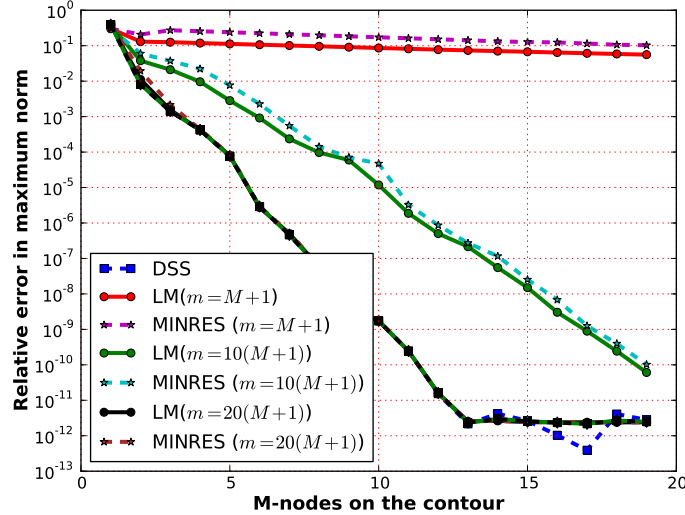


Figure 4.4. Relative error in the maximum norm vs. number of nodes on the contour for different number of iterations,  $m$ , in solving the 2D heat equation (2.4). The Lanczos and the minimal residual methods result in a very high relative error as compared to the direct sparse solver for  $m = M + 1$ , but they have the same relative error as the direct sparse solver when increasing the number of iterations to  $m = 20(M + 1)$ .

As we see from Figure 4.4, the LM and MINRES methods result in a low accuracy as compared to the DSS for small number of iterations, namely  $m = M + 1$ , in the Lanczos process. We experiment by increasing the number of iterations in the Lanczos process and we observe, as expected, the error decreases as we increase the number of iterations. In our experiment, when we take  $m = 20(M + 1)$  in LM and MINRES we get the same relative error as DSS as shown Figure 4.4. However, the storage requirement and computational time increase as the number of iterations in the Lanczos process increases.

Next, we compare the relative error versus the number of nodes on the contour of the preconditioned methods, PrecLM and PrecMINRES, with DSS for  $m = (M + 1)$ . As Figure 4.5 shows unlike the LM and MINRES the PrecLM and PrecMINRES give the same relative error as DSS for  $m = M + 1$ . Therefore, the preconditioned methods yield the same relative accuracy as DSS without additional memory space as compare to the LM and MINRES with  $m = 20(M + 1)$ .

Finally, we compare the relative error versus CPU-time of all the methods for the symmetric



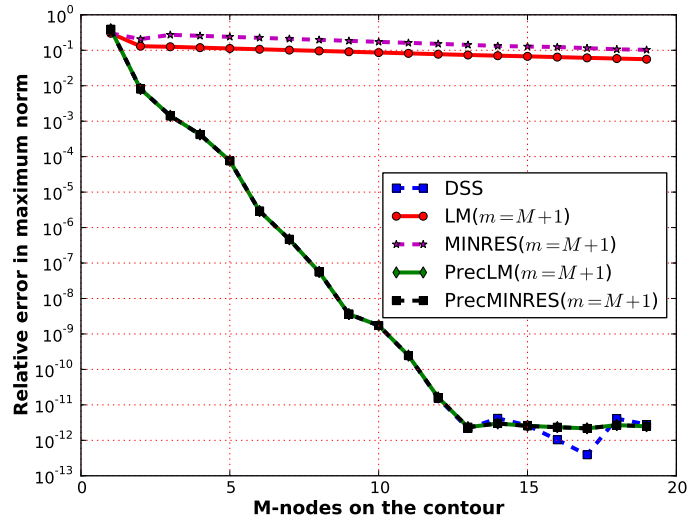


Figure 4.5. *Relative error in the maximum norm vs. number of nodes on the contour for different number of iterations,  $m$ , in solving the symmetric problem (2.4) using preconditioning. By preconditioning the Lanczos method and the minimal residual method we achieve the same accuracy as the direct sparse solver for a relatively small number of iterations ( $m = M + 1$ ).*

problem with DSS as shown in Figure 4.6. These numerical results show that the PrecLM and PrecMINRES require less computation time as compared to LM, MINRES and DSS.

In conclusion, the above numerical experiments show that the preconditioned methods, PrecLM and PrecMINRES, with  $m = M + 1$  require less memory space and computational time as compare to the LM and MINRES with  $m = 20(M + 1)$ . In addition, they result in the same accuracy as the DSS, which gives the exact solution in the absence of roundoff errors. Therefore the preconditioning improves the efficiency of the these iterative methods. Next, we experiment with the nonsymmetric problem (2.5).

## 4.5.2 Nonsymmetric Problem

We do similar numerical experiments as the above on the two-dimensional heat equation with convection term (2.5) to compare FOM, GMRES and DSS.

Figure 4.7 shows that FOM and GMRES result in a significantly lower accuracy as compare to the DSS for  $m = M + 1$ . In addition, by taking larger  $m = 20(M + 1)$  we get the same

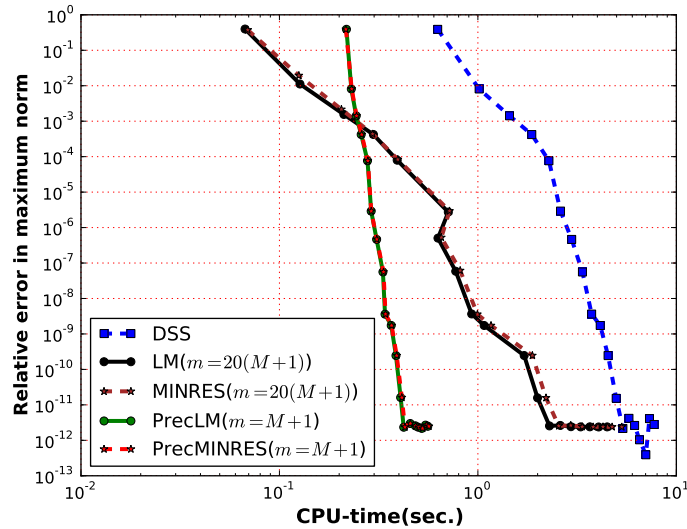


Figure 4.6. *Relative error in the maximum norm vs. CPU-time in solving the symmetric problem (2.4). The preconditioned Lanczos and minimal residual methods with  $m = M + 1$  require less computation time as compared to the unpreconditioned Lanczos and minimal residual methods with  $m = 20(M + 1)$  and the direct sparse solver.*

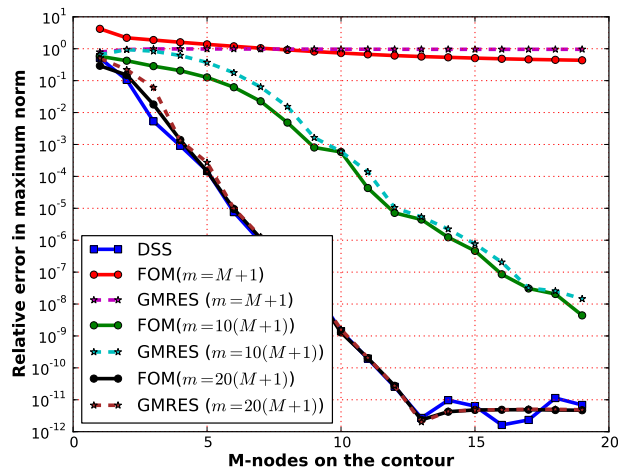


Figure 4.7. *Relative error in the maximum norm vs. number of nodes on the contour for different  $m$  in solving the nonsymmetric problem (2.5). The full orthogonalization method and general minimal residual method without preconditioning have a very high relative error for  $m = M + 1$  as compared to the direct sparse solver, but increasing the iteration number to  $m = 20(M + 1)$  results in the same accuracy.*

relative error as DSS as shown in the same Figure 4.7. However, the computation time and the memory space increase when we take a large number of iterations in the Arnoldi process.

We now compare the relative error versus the number of iterations in the Arnoldi process of the preconditioned methods, PrecFOM and PrecGMRES, with DSS. As we can see from

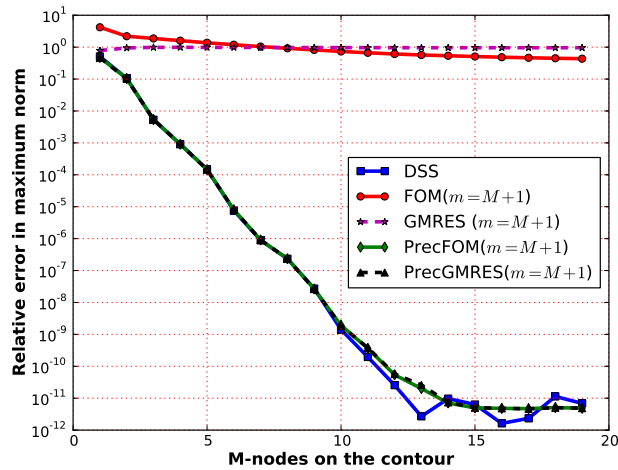


Figure 4.8. *Relative error in the maximum norm vs. number of nodes on the contour in solving the nonsymmetric problem (2.5) by preconditioning. Preconditioning the full orthogonalization method and the minimal residual method yields the same accuracy as the direct sparse solver for  $m = M + 1$ .*

Figure 4.8 for  $m = (M + 1)$  the PrecFOM and PrecGMRES give the same relative error as DSS. Therefore, unlike the FOM and GMRES the PrecFOM and PrecGMRES have the same accuracy as the DSS for  $m = (M + 1)$ . In order to investigate which method is more efficient we need to compare computational times.

We finally compare the relative error versus CPU-time of FOM, GMRES, PrecFOM, PrecGMRES and DSS as shown in Figure 4.8. These numerical results show that the PrecFOM and PrecGMRES with  $m = M + 1$  require less computation time than FOM and GMRES with  $m = 20(M + 1)$  and DSS.

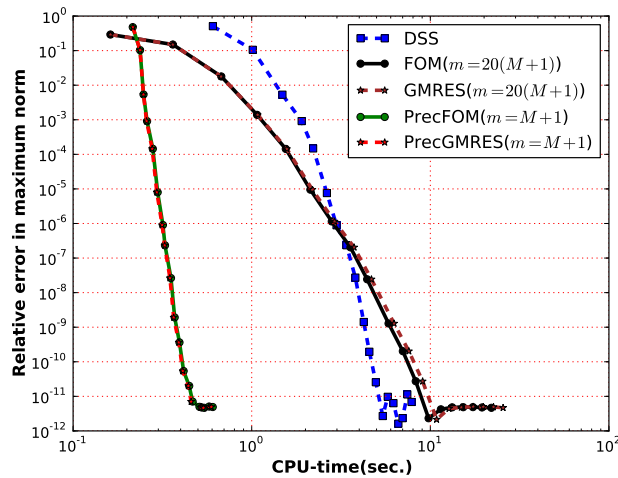


Figure 4.9. *Relative error in the maximum norm vs. CPU-time for solving (2.5) in comparison of the preconditioned and unpreconditioned methods. The CPU-time of the unpreconditioned method with  $m = 20(N + 1)$  is large as compared to the direct sparse solver for small relative error. In addition, the preconditioned methods with a relatively small number of iterations ( $m=M+1$ ) require less CPU-time as compared to the other methods.*

## 4.6 Summary and conclusion

In this chapter we discussed the Krylov subspace methods for solving symmetric and nonsymmetric shifted sparse linear systems of large order. For the sparse system we did not apply the Hessenberg reduction as we did for the dense system since Householder reflection will destroy the sparsity of the matrix. Instead, we reduced the large sparse system to a much smaller system using Krylov subspace methods. Since Krylov subspace methods converge slowly we preconditioned the system so that they converge faster. We discussed a shift-invert preconditioner for the shifted linear systems and showed how it improves the convergence when an appropriate shift parameter in the preconditioner is chosen.

We also compared the direct sparse solver with the orthogonal projection methods, LM and FOM, for the symmetric system and the minimal residual methods, MINRES and GMRES, for the nonsymmetric system. We found that the preconditioned methods require low computational times and yield the same accuracy as the direct sparse solver using a Krylov subspace of a relatively small dimension. This also implies that the memory requirement of these preconditioned methods is small as compared to the direct sparse solver.

# Appendix

## Improved Hessenberg function

We modified the built-in hessenberg function in SciPy version 0.7.0 to improve the explicit computation of the orthogonal matrix. We used the matrix-vector function `gemv` instead of matrix-matrix function `gemm` and backward accumulation for a better efficiency [9, p. 213].

Listing 1: Improved hessenberg function

```
'''
We modify the built-in hessenberg function in SciPy version 0.7.0 to
improve the explicit computation of the orthogonal matrix. We use the
matrix-vector product function 'gemv' instead of the matrix-matrix
product function 'gemm' and backward accumulation for efficiency. For
backward accumulation see Golub & Van Loan (3rd), p. 213.
'''

def hessenberg(a, calc_q=0, overwrite_a=0):
    a1 = asarray(a)
    if len(a1.shape) != 2 or (a1.shape[0] != a1.shape[1]):
        raise ValueError, 'expected square matrix'
    overwrite_a = overwrite_a or (_datanotshared(a1, a))

    gehrd, gebal = get_lapack_funcs(('gehrd', 'gebal'), (a1,))
    ba, lo, hi, pivscale, info = gebal(a, permute=1,
                                       overwrite_a=overwrite_a)
    if info < 0:
        raise ValueError, 'illegal value in %d-th argument of internal gebal
                           (hessenberg)' % (-info)
    n = len(a1)
    lwork = calc_lwork.gehrd(gehrd.prefix, n, lo, hi)
    hq, tau, info = gehrd(ba, lo=lo, hi=hi, lwork=lwork, overwrite_a=1)
    if info < 0:
        raise ValueError, 'illegal value in %d-th argument of internal gehrd
                           (hessenberg)' % (-info)
    if not calc_q:
        for i in range(lo, hi):
```

```
        hq[i+2:hi+1,i] = 0.0
    return hq
# XXX: Use ORGHR routines to compute q.
# The above lines of codes do not change. We only modify the following
# code
# for the explicit computation of q.
# E.B. Eneyew, December 2010.
ger, gemv = get_blas_funcs(('ger', 'gemv'), (hq,))
typecode = hq.dtype.char
q = None
for i in range(hi-1, lo-1, -1):
    if tau[i] == 0:
        continue
    v = zeros(n, dtype=typecode)
    v[i+1] = 1.0
    v[i+2:hi+1] = hq[i+2:hi+1, i]
    hq[i+2:hi+1, i] = 0.0
    if q == None:
        q = diag(ones(n, dtype=typecode))
    w = gemv(1, q, v, beta=0, y=None, offx=0, incx=1, offy=0,
            incy=1, trans=2)
    q = ger(-tau[i], v, w, a=q, overwrite_a = 1)
if q == None:
    q = diag(ones(n, dtype=typecode))
return hq, q
```

\*\*\*\*\*

# Bibliography

- [1] R. Barrett, M. Berry, T.F. Chan, and et al., *Templates for the solution of linear systems: building blocks for iterative methods*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1994.
- [2] M. Benzi, *Preconditioning techniques for large linear systems: a survey*, J. Comput. Phys. **182** (2002), no. 2, 418–477.
- [3] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang, *Spectral methods*, Springer-Verlag, Berlin, 2006.
- [4] SciPy community, *SciPy Reference Guide*, 2010, Available from <http://docs.scipy.org/doc/scipy/reference/index.html>.
- [5] T. A. Davis, *Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method*, ACM Trans. Math. Software **30** (2004), no. 2, 196–199.
- [6] ———, *Direct methods for sparse linear systems*, Fundamentals of Algorithms, vol. 2, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2006.
- [7] J. W. Demmel, *Applied numerical linear algebra*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.
- [8] D. G. Duffy, *Advanced engineering mathematics with MATLAB®*, CRC Press, Boca Raton, FL, 2011, Third edition.
- [9] G. H. Golub and C. F. Van Loan, *Matrix computations*, third ed., Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, 1996.

- 
- [10] J. Grotendorst, D. Marx, and A. Muramatsu, *Quantum simulations of complex many body systems from theory to algorithms*, lecture notes ed., John von Neumann Institute for computing, NIC-secretariat, Research center Jülich, 52425 Jülich, Germany, 2002.
- [11] K. J. in 't Hout and J. A. C. Weideman, *A contour integral method for the Black-Scholes and Heston equations*, SIAM J. Sci. Comput. **33** (2011), no. 2, 763–785.
- [12] A. Iserles, *A first course in the numerical analysis of differential equations*, second ed., Cambridge Texts in Applied Mathematics, Cambridge University Press, Cambridge, 2009.
- [13] Z. Jia, *The convergence of Krylov subspace methods for large unsymmetric linear systems*, Acta Math. Sinica (N.S.) **14** (1998), no. 4, 507–518.
- [14] A.K. Kassam and L. N. Trefethen, *Fourth-order time-stepping for stiff PDEs*, SIAM J. Sci. Comput. **26** (2005), no. 4, 1214–1233.
- [15] J. Liesen and P. Tichý, *Convergence analysis of Krylov subspace methods*, GAMM Mitt. Ges. Angew. Math. Mech. **27** (2004), no. 2, 153–173 (2005).
- [16] K. Meerbergen, *The solution of parametrized symmetric linear systems*, SIAM J. Matrix Anal. Appl. **24** (2003), no. 4, 1038–1059.
- [17] C. Moler and C. Van Loan, *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Rev. **45** (2003), no. 1, 3–49 (electronic).
- [18] A. Neumaier, *Introduction to numerical analysis*, Cambridge University Press, Cambridge, 2001.
- [19] P. V. O’Neil, *Advanced engineering mathematics*, fifth ed., Brooks/Cole-Thomson, Pacific Grove, CA, 2003.
- [20] Y. Saad, *Iterative methods for sparse linear systems*, second ed., Dover Publications Inc., Mineola, NY, 2000.
- [21] V. Simoncini and D. B. Szyld, *Recent computational developments in Krylov subspace methods for linear systems*, Numer. Linear Algebra Appl. **14** (2007), no. 1, 1–59.
- [22] L.N. Trefethen, *Spectral methods in MATLAB*, Software, Environments, and Tools, vol. 10, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.



- 
- [23] L.N. Trefethen and D. Bau, III, *Numerical linear algebra*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.
- [24] D. S. Watkins, *Fundamentals of matrix computations*, Pure and Applied Mathematics (New York), Wiley-Interscience [John Wiley & Sons], New York, 2002, Second edition.
- [25] J. A. C. Weideman and S. C. Reddy, *A MATLAB differentiation matrix suite*, ACM Trans. Math. Software **26** (2000), no. 4, 465–519.
- [26] J. A. C. Weideman and L. N. Trefethen, *Parabolic and hyperbolic contours for computing the Bromwich integral*, Math. Comp. **76** (2007), no. 259, 1341–1356.