INCORPORATING KRYLOV SUBSPACE METHODS IN THE ETDRK4 SCHEME

by

Jeffrey H. Allen

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in

Mathematics

at

The University of Wisconsin–Milwaukee

May 2014

Abstract

INCORPORATING KRYLOV SUBSPACE METHODS IN THE ETDRK4 SCHEME

by

Jeffrey H. Allen

The University of Wisconsin–Milwaukee, 2014
Under the Supervision of Professor Bruce Wade

A modification of the $(2, 2)$-Padé algorithm developed by Wade et al. for implementing the exponential time differencing fourth order Runge-Kutta (ETDRK4) method is introduced. The main computational difficulty in implementing the ETDRK4 method is the required approximation to the matrix exponential. Wade et al. use the fourth order $(2, 2)$-Padé approximant in their algorithm and in this thesis we incorporate Krylov subspace methods in an attempt to improve efficiency. A background of Krylov subspace methods is provided and we describe how they are used in approximating the matrix exponential and how to implement them into the ETDRK4 method. The $(2, 2)$-Padé and Krylov subspace algorithms are compared in solving the one and two dimensional Allen-Cahn equation with the ETDRK4 scheme. We find that in two dimensions, the Krylov subspace algorithm is faster, provided we have a spatial discretization that produces a symmetric matrix.

To my family

Table of Contents

List of Figures

Acknowledgements

First I would like to thank Prof. Bruce Wade for his constant words of encouragement and help throughout the past months during the research and writing process. I have learned a lot under his guidance and I am very grateful that I had the opportunity to acquaint myself with such fascinating problems.

I also want to thank my mother and father for their continuous support and for always believing in me. Talking to them often fostered my dedication and motivation towards this thesis.

Milwaukee, Wisconsin Jeffrey Allen

May, 2014

Chapter 1

Introduction

We are concerned with solving systems of stiff nonlinear reaction diffusion equations using a special class of time-stepping methods known as Exponential Time Differencing (ETD) schemes. ETD schemes have been around since the 1960's, but only in the past ten years has there been a resurgence in interest due to some minor breakthroughs in methods of implementation [6],[7]. The Runge-Kutta versions (ETDRK) of these schemes compete with other stiff solvers such as integrating factor and linearly implicit schemes [6]. These ETDRK methods were developed, in part, by Cox and Matthews [1] and have drawn considerable research interest due to their ability to solve stiff systems of ODE's without requiring prohibitively small time steps. The method we focus on here is the ETDRK4 scheme, a fourth order Runge-Kutta method developed by Cox and Matthews in [1].

As we will see, though, computing solutions from the ETDRK4 formulae directly can be numerically unstable. Cox and Matthews were aware of the difficulties in solving linear systems coming from the formulae, and Kassam and Trefethen found a way around this numerical instability using contour integration in [7]. However, even medium sized problems can render this method unrealistic computationally because of excessively large contours of integration.

In [14], direct computation of expressions containing the matrix exponential, which are the root of computational difficulties, is avoided by the use a fourth order diagonal Padé approximation to the matrix exponential. The Padé approximant can be expanded in a partial fraction decomposition and obtaining the desired quantities for the solution now only requires solving a few linear systems for each time step.

An additional focus of that paper is dealing with non smooth initial data by using a slightly different Padé approximant to the matrix exponential that smooths out spurious oscillations. We will not continue with that direction here, but the diagonal Padé approximation will be explored further as it is the foundation of the proposed scheme, which relies on Krylov subspace methods.

Krylov subspace methods have become a popular tool in the implementation of exponential integrators [5],[6]. This popularity is due to the ubiquity of the matrix exponential, and Krylov subspace methods attempt to improve the efficiency in its computation by exploiting the sparsity commonly found in matrices arising from spatial discretizations [2]. We are unaware of any research that implements Krylov subspace methods into the ETDRK4 scheme. Implementing Krylov subspace methods into ETDRK4 and experimentally comparing efficiency with the diagonal Padé scheme is the focus of this paper.

Chapter 2 consists of background information where we will introduce the class of PDE we will be solving and where we derive the ETD schemes. The ETDRK4 scheme will be discussed in further detail and we will also describe some methods of computation. In Chapter 3 we will introduce the theory of Krylov subspace methods and the relevant algorithms. Then in Chapter 4 we will present numerical experiments that reveal the computational differences between the Krylov subspace and Padé approximation methods of implementing ETDRK4.

Chapter 2

Background and ETD Schemes

We restrict our attention to solving reaction diffusion systems, which, in the current setting, are manifested in nonlinear parabolic partial differential equations. Deriving the ETD schemes requires first discretizing the spatial variable of this PDE. Finite differences, finite elements, and spectral methods are among the various types of discretizations that are possible here. We will arrive at a time stepping scheme from which several different ETD schemes of various orders have been developed. The ETDRK4 is one of these schemes and it will be introduced and discussed in detail. Its computation will be the focus of the rest of this chapter.

2.1 The Reaction Diffusion System

The reaction diffusion system gives the following nonlinear initial-boundary value problem:

$$\begin{aligned} u_t + Au &= F(u,t) \qquad \text{in } \Omega, \qquad t \in \left(0, \bar{t}\ \right] = J, \\ u &= v \qquad \text{on } \partial\Omega, \quad t \in J, \quad u(\cdot,0) = u_0 \quad \text{in } \Omega, \end{aligned} \tag{2.1}$$

Here, $\Omega \subset \mathbb{R}^d$ is bounded with Lipschitz continuous boundary, A is an uniformly elliptic operator, and $F \in C^1(\mathbb{R}^{d+1}, \mathbb{R}^d)$ is typically nonlinear.

We will derive the foundation of the ETD schemes in an abstract framework.

Assume that A is a partial differential operator that takes the form

$$A := -\sum_{j,k=1}^{d} \frac{\partial}{\partial x_j}\left(a_{j,k}(x)\frac{\partial}{\partial x_k}\right) + \sum_{j=1}^{d} b_j(x)\frac{\partial}{\partial x_j} + b_0(x),$$

where $a_{j,k}, b_j \in C^\infty(\overline{\Omega})$ and $a_{j,k} = a_{k,j}$, $b_0 \geq 0$. The uniform ellipticity of A means that for some $c_0 > 0$ we have

$$\sum_{j,k=1}^{d} a_{j,k}(\cdot)\xi_j\xi_k \geq c_0|\xi|^2, \quad \text{on } \overline{\Omega}, \quad \text{for all } \xi \in \mathbb{R}^d.$$

In order to simplify the analysis, we will work in a general Hilbert space X rather than $\mathbb{R}^d$. Now we can consider the operator A to be a linear, self-adjoint, positive definite and closed operator with a compact inverse T, defined on a dense domain $D(A) \subset X$. The operator A will usually represent some spatial discretization of Ω.

We assume the resolvent set $\rho(A)$ of A satisfies, for some $\alpha \in (0, \frac{\pi}{2})$,

$$\rho(A) \supset \overline{\Sigma}_\alpha, \quad \Sigma_\alpha := \{z \in \mathbb{C} : \alpha < |\arg(z)| \leq \pi, z \neq 0\}.$$

This is the assumption that the eigenvalues of A have negative real part. Also, assume there exists $M \geq 1$ such that

$$\|(zI - A)^{-1}\| \leq M|z|^{-1}, \qquad z \in \Sigma_\alpha.$$

The norm $\|\cdot\|$ will denote the matrix and vector 2-norm throughout this thesis. Therefore we get that $-A$ is the infinitesimal generator of an analytic semigroup $\{e^{-tA}\}_{t\geq 0}$ which is the solution operator for (2.2) below, [14]. The standard representation is

$$E(t) := e^{-tA} = \frac{1}{2\pi i}\int_\Lambda e^{-tz}(zI - A)^{-1}dz,$$

where $\Lambda := \{z \in \mathbb{C} : |\arg(z)| = \theta\}$ is oriented so that $\mathrm{Im}(z)$ decreases, for any

$\theta \in (\alpha, \frac{\pi}{2})$.

Using the Duhamel principle, we can write the exact solution of (2.1) as

$$u(t) = E(t)v + \int_0^t E(t-s)F(u(s),s)ds. \tag{2.2}$$

Let $0 < k \leq k_0$, for some k_0, and $t_n = nk$, $0 \leq n \leq N$. Replacing t by $t+k$, using basic properties of E and by the change of variable $s - t = k\tau$, we can arrive at

$$u(t+k) = E(k)u(t) + k\int_0^1 E(k-k\tau)F(u(t+k\tau), t+k\tau)d\tau,$$

which satisfies the recurrence formula

$$u(t_{n+1}) = e^{-kA}u(t_n) + k\int_0^1 e^{-kA(1-\tau)}F(u(t_n+\tau k), t_n+\tau k)\,d\tau. \tag{2.3}$$

2.2 ETDRK4

Equation (2.3) is exact and the various ETD schemes come from how one approximates the integral and the matrix exponential. From now on, when we write e^{-kA}, A is a matrix defined on n-dimensional Euclidean space; we will not be concerned with operators defined on infinite dimensional function spaces. Cox and Matthews developed the fourth order scheme ETDRK4 by approximating the integral with the classical fourth order Runge-Kutta approximation, but A is restricted to a very limited class of matrices. The $(n+1)^{\text{st}}$ approximation to the solution is given by

$$\begin{aligned} u_{n+1} = {} & e^{-kA}u_n + f_1(kA)F(u_n, t_n) \\ & + 2f_2(kA)\big(F(a_n, t_n + k/2) + F(b_n, t_n + k/2)\big) \\ & + f_3(kA)F(c_n, t_n + k) \end{aligned} \tag{2.4}$$

where

$$\begin{aligned} a_n &= e^{-kA/2}u_n - A^{-1}(e^{-kA/2} - I)F(u_n, t_n) \\ b_n &= e^{-kA/2}u_n - A^{-1}(e^{-kA/2} - I)F(a_n, t_n + k/2) \\ c_n &= e^{-kA/2}a_n - A^{-1}(e^{-kA/2} - I)\big(2F(b_n, t_n + k/2) - F(u_n, t_n)\big) \end{aligned} \tag{2.5}$$

and

$$\begin{aligned} f_1(kA) &= k^{-2}(-A)^{-3}\Big[-4 + kA + e^{-kA}(4 + 3kA + k^2A^2)\Big] \\ f_2(kA) &= k^{-2}(-A)^{-3}\Big[2 - kA + e^{-kA}(-2 - kA)\Big] \\ f_3(kA) &= k^{-2}(-A)^{-3}\Big[-4 + 3kA - k^2A^2 + e^{-kA}(4 + kA)\Big]. \end{aligned} \tag{2.6}$$

Deriving these formulas is nontrivial and is aided by Maple [7]. The formulas are problematic, however, because of cancellation errors coming from expressions of the form

$$\varphi(z) = \frac{e^{-z} - 1}{z} \tag{2.7}$$

when z is close to zero. The matrix analogue for this expression is $A^{-1}(e^{-A} - I)$ and this term suffers from cancellation error when the eigenvalues of A are close to zero [1]. The formulas for f_1, f_2, and f_3 contain higher order versions of these terms and these suffer from even further instability if the eigenvalues of A are close to zero. Cox and Matthews were aware of this and in [1] they restrict their attention to matrices whose eigenvalues are at least some distance away from zero. The matrices used in this thesis will come from spatial discretizations and will invariably have eigenvalues that are close to zero.

Trefethen and Kassam find a way around this using Cauchy's integral formula from complex analysis. Given an open set $U \subset \mathbb{C}$, a holomorphic function $f : U \to$

$\mathbb{C}$, and $z \in \mathbb{C}$ on the interior of a contour Γ, we have

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

and the matrix analogue for this formula is

$$f(A) = \frac{1}{2\pi i} \int_\Gamma f(\zeta)(\zeta I - A)^{-1}\, d\zeta$$

where Γ is now a contour enclosing the eigenvalues of A.

This method of computing quantities in (2.4) and (2.5) is accurate, but for problems with large spectral radii this method is unrealistic since the contour of integration must encircle the spectrum [14]. Since the matrix A will come from some spatial discretization, such as finite differences, the eigenvalues of A will tend towards infinity as the size of the problem increases. That is, for increasingly finer discretizations, we need to integrate over ever larger contours and this can become an intractable computation for large problems in multiple dimensions.

When the matrix A is large, a more conspicuous problem in (2.4-6) is computing the matrix exponential e^{-kA}. Kassam and Trefethen do not acknowledge this in their paper and in their provided code they simply use Matlab's `expm` function. This will be problematic because this function has $O(n^3)$ complexity, which will be too slow for large problems. In [7] they do, however, acknowledge that large problems in multiple dimensions could render their method computationally unrealistic.

In [14], directly computing the matrix exponential is avoided by using a rational approximation to e^{-kA}. Specifically, they use the (2,2)-Padé approximant to e^{-kA} denoted here by $R_{2,2}(kA)$ and given by

$$R_{2,2}(kA) = (12I - 6kA + k^2A^2)(12I + 6kA + k^2A^2)^{-1}.$$

This approximant is fourth order in the sense that $\|e^{-kA} - R_{2,2}(kA)\| \leq Ck^4$, where C can depend on A. Utilizing this approximant in (2.4-6), we get

$$\begin{aligned} u_{n+1} &= R_{2,2}(kA)u_n + P_1(kA)F(u_n, t_n) \\ &\quad + P_2(kA)\Big(F(a_n, t_n + k/2) + F(b_n, t_n + k/2)\Big) + P_3(kA)F(c_n, t_n + k), \end{aligned} \tag{2.8}$$

where

$$\begin{aligned} P_1(kA) &= k(2I - kA)(12I + 6kA + k^2A^2)^{-1}, \\ P_2(kA) &= 4k(12I + 6kA + k^2A^2)^{-1}, \\ P_3(kA) &= k(2I + kA)(12I + 6kA + k^2A^2)^{-1}. \end{aligned}$$

We use the $(2,2)$-Padé approximant to $e^{-kA/2}$ denoted by $\widetilde{R}_{2,2}(kA)$, as follows:

$$\begin{aligned} a_n &= \widetilde{R}_{2,2}(kA)u_n + \widetilde{P}(kA)F(u_n, t_n), \\ b_n &= \widetilde{R}_{2,2}(kA)u_n + \widetilde{P}(kA)F(a_n, t_n + k/2), \\ c_n &= \widetilde{R}_{2,2}(kA)a_n + \widetilde{P}(kA)\Big(2F(b_n, t_n + k/2) - F(u_n, t_n)\Big), \end{aligned}$$

with

$$\begin{aligned} \widetilde{R}_{2,2}(kA) &= (48I - 12kA + k^2A^2)(48I + 12kA + k^2A^2)^{-1}, \\ \widetilde{P}(kA) &= 24k(48I + 12kA + k^2A^2)^{-1}. \end{aligned}$$

In order to compute a_n, b_n, and c_n as displayed above, we won't actually compute the matrix $\widetilde{R}_{2,2}(kA)$, but instead use the partial fraction decomposition. This way inverting cubic and quadratic matrix polynomials is avoided, which would be numerically unstable and computationally burdensome. Instead, we are left with a few linear systems involving the matrix A. Here are the partial fraction decompositions

of the required Padé approximations. To compute u_{n+1}, we will utilize

$$R_{2,2}(z) = 1 + 2\,\Re\left(\frac{w_1}{z - c_1}\right)$$

and the corresponding $\{P_i(z)\}_{i=1}^{3}$ takes the form

$$P_i(z) = 2k\,\Re\left(\frac{w_{i1}}{z - c_1}\right), \quad i = 1, 2, 3$$

where $R_{2,2}$ and P_i have the complex poles c_1 and $\bar{c}_1$, with w_1,w_{i1} the corresponding weights for $i = 1, 2, 3$.

To compute a_n, b_n, and c_n, we use that

$$\widetilde{R}_{2,2}(z) = 1 + 2\,\Re\left(\frac{\widetilde{w}_1}{z - \widetilde{c}_1}\right)$$

and the corresponding $\widetilde{P}(z)$ as

$$\widetilde{P}(z) = 2k\,\Re\left(\frac{\widetilde{\Omega}_1}{z - \widetilde{c}_1}\right),$$

where $\widetilde{R}_{2,2}$ and $\widetilde{P}$ have the complex poles $\widetilde{c}_1$ and $\overline{\widetilde{c}}_1$. The corresponding weights for $\widetilde{R}_{2,2}$ and $\widetilde{P}$ are $\widetilde{w}_1$ and $\widetilde{\Omega}_1$, respectively. The parallel $(2,2)$-Padé algorithm can now be stated [14]:

The $(2,2)$-Padé ETDRK4 Algorithm

1. To compute a_n, solve

$$(kA - \widetilde{c}_1 I)Na_1 = \widetilde{w}_1 u_n + k\,\widetilde{\Omega}_1 F(u_n, t_n),$$

for Na_1 and then

$$a_n = u_n + 2\,\Re\,(Na_1)$$

2. To compute b_n, solve

$$(kA - \widetilde{c}_1 I)Nb_1 = \widetilde{w}_1 u_n + k\,\widetilde{\Omega}_1 F(a_n, t_n + k/2),$$

for Nb_1 and then

$$b_n = u_n + 2\,\Re\,(Nb_1)$$

3. Similarly, to compute c_n, solve

$$(kA - \widetilde{c}_1 I)Nc_1 = \widetilde{w}_1 a_n + k\,\widetilde{\Omega}_1 \Big(2F(b_n, t_n + k/2) - F(u_n, t_n)\Big),$$

for Nc_1 and then

$$c_n = a_n + 2\,\Re\,(Nc_1)$$

4. Finally, to compute u_{n+1}, first solve

$$\begin{aligned}(kA - c_1 I)Nu_1 = {} & w_1 u_n + k w_{11} F(u_n, t_n) \\ & + k w_{21}\Big(F(a_n, t_n + k/2) + F(b_n, t_n + k/2)\Big) \\ & + k w_{31} F(c_n, t_n + k),\end{aligned}$$

for Nu_1 and then compute

$$u_{n+1} = u_n + 2\,\Re\,(Nu_1)\,.$$

The poles and weights are fixed once and for all [14]:

$$
\begin{aligned}
c_1 &= -3.0 + i\,1.73205080756887729352,\\
w_1 &= -6.0 - i\,10.3923048454132637611,\\
w_{11} &= -0.5 - i\,1.44337567297406441127,\\
w_{21} &= -i\,1.15470053837925152901,\\
w_{31} &= 0.5 + i\,0.28867513459481288225,\\
\widetilde{c}_1 &= -6.0 + i\,3.4641016151377545870548,\\
\widetilde{w}_1 &= -12.0 - i\,20.7846096908265275223293 5,\\
\widetilde{\Omega}_1 &= -i\,3.4641016151377545870 5.
\end{aligned}
$$

Chapter 3

Krylov Subspace Methods

3.1 Background

Computing the Matrix Exponential

The most computationally difficult component of the ETDRK4 is the matrix exponential e^{-kA}: computing it efficiently and accurately has been a challenge for decades. The 2003 update-republication of Moler's and Van Loan's article *Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later* supports this [8]. In the original article from 1978 there was no mention of Krylov subspace methods, but the 2003 update supplied a few more methods and the Krylov subspace methods were among these.

Lying at the base of many methods is the (p, q)-Padé approximation to e^{-kA} denoted here by $R_{p,q}(kA)$. Khaliq et. al. use the (2,2)-Padé approximant because it is all that is required to maintain a method that is fourth order in space. However, for general computations, p and q are required to be larger for the sake of accuracy.

The most popular method of computing the matrix exponential today is the scaling and squaring method [8]. Indeed, this is the algorithm that Matlab's `expm` function uses to approximate the matrix exponential. At its core, scaling and squaring relies on the Padé approximation, but it first uses a fundamental identity of the exponential function in order to circumvent roundoff error otherwise accumulated in the Padé approximation. The method relies on the elementary property

$$e^A = (e^{A/m})^m.$$

This property becomes useful when one realizes that computing the matrix exponential with Padé approximants is prone to roundoff error when $\|A\|$ is large. The scaling and squaring method chooses m to be the smallest power of 2 such that $\|A/m\| \leq c$, where c depends on the problem and should be small in general. Now $e^{A/m}$ can be computed in the absence of detrimental roundoff error with Padé approximants, and this matrix is then repeatedly squared to give $e^A = (e^{A/m})^m$ [8]. The authors of [7] use this method to compute e^A and $e^{A/2}$. The scaling and squaring method requires $O(n^3)$ operations and for very large matrices this is prohibitively time consuming.

In (2.4-6) we notice that every appearance of the matrix exponential is accompanied only by its action on a vector. Ideally, we would never have to compute the full matrix exponential, only its *action* on the vector. This is precisely what is accomplished by using Krylov subspace methods. We will now provide a background of these methods.

Krylov Subspaces

The development of Krylov subspace methods in the 1950's was motivated by the need for faster algorithms that approximated eigenvalues of a matrix [13]. There are two basic iterative algorithms that comprise Krylov subspace methods: the *Arnoldi iteration* for general matrices and the *Lanczos iteration* for symmetric matrices. Both iterations are able to approximate the extremal eigenvalues in $O(n^2)$ operations, and they were later adapted as iterative methods for solving linear systems. The Lanczos iteration was applied to approximating the solution to the symmetric, positive definite system $Ax = b$, a procedure now known as the *Conjugate Gradient Method*. The Arnoldi method was similarly applied to obtain the *GMRES method* for solving general linear systems $Ax = b$. Since all of the matrices considered in this thesis will be symmetric, we will restrict most of our attention to the Lanczos iteration.

Recall that for a matrix $A \in \mathbb{R}^{n\times n}$, we have the Hessenberg decomposition $Q^TAQ = H$ where $Q \in \mathbb{R}^{n\times n}$ is orthogonal and $H \in \mathbb{R}^{n\times n}$ is upper Hessenberg, meaning it is upper triangular with possible nonzero entries on its first subdiagonal. The algorithm that computes this decomposition uses Householder matrices and involves $O(n^3)$ operations. For a *symmetric* matrix $A \in \mathbb{R}^{n\times n}$, we have the *tridiagonal* decomposition $Q^TAQ = T$ where $Q \in \mathbb{R}^{n\times n}$ is orthogonal and $T \in \mathbb{R}^{n\times n}$ is tridiagonal. Similarly, the algorithm that computes this decomposition also uses Householder matrices and requires about half as many operations as the Hessenberg reduction if symmetry is exploited [13].

There is a connection between Krylov subspace methods with these decompositions that will become apparent after we define the Krylov subspaces. Given a matrix $A \in \mathbb{R}^{n\times n}$ and a vector $v \in \mathbb{R}^n$, define the mth *Krylov subspace* $\mathcal{K}_m(A, v)$ as

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

Thus, $\mathcal{K}_m(A, v)$ is the range of the mth *Krylov matrix* $K_m(A, v) \in \mathbb{R}^{n\times m}$

$$K_m(A, v) = \begin{pmatrix} | & | & & | \\ v & Av & \cdots & A^{m-1}v \\ | & | & & | \end{pmatrix}$$

The connection between Krylov subspaces and the tridiagonal decomposition of a symmetric matrix involves the QR factorization of the Krylov matrix $K_n(A, v)$ [13]. More precisely, we have the following fact: if $Q^TAQ = T$ is the tridiagonal decomposition of the symmetric matrix $A \in \mathbb{R}^{n\times n}$, then $Q^TK_n(A, q_1) = R$ is upper triangular, where q_1 is the first column of Q [4]. The matrix R is the Krylov matrix $K_n(T, e_1)$ where e_1 is the first element of the canonical basis of $\mathbb{R}^n$.

The Lanczos Iteration

Let us now see how we can compute the matrix T. Suppose that Q has columns $q_1, \ldots, q_n$ and that

$$T = \begin{pmatrix} \alpha_1 & \beta_1 & & \cdots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{n-1} \\ 0 & \cdots & & \beta_{n-1} & \alpha_n \end{pmatrix}$$

Equating the columns of the equation $AQ = QT$, we have the relation

$$Aq_k = \beta_{k-1}q_{k-1} + \alpha_k q_k + \beta_k q_{k+1} \qquad \text{for } k = 1, \ldots, n-1 \tag{3.1}$$

where we take $\beta_0 q_0 = 0$. Since the columns of Q are orthonormal, premultiplying both sides of (3.1) by q_k^T yields $\alpha_k = q_k^T A q_k$. If $r_k = \beta_k q_{k+1} = Aq_k - \alpha_k q_k - \beta_{k-1}q_{k-1}$, then $q_{k+1} = r_k/\beta_k$ and $\beta_k = \pm\|r_k\|$ [4]. The matrices Q and T are created using this recurrence relation, and it is called the *Lanczos iteration* after its inventor. If one only needs the matrix T, then q_k are overwritten and the matrix Q is never explicitly formed because it would require significant storage if A is large. This recurrence can be carried out for $k = 1, \ldots, m$ with $m < n-1$ to obtain the partial tridiagonalization

$$AV_m = V_m T_m + \beta_m v_{m+1} e_m^T \tag{3.2}$$

where $V_m \in \mathbb{R}^{n\times m}$ has orthonormal columns $v_1, \ldots, v_m$, $T_m \in \mathbb{R}^{m\times m}$ is tridiagonal and symmetric, and e_m is the last element of the canonical basis for $\mathbb{R}^m$ [13]. The columns of V_m are called the *Lanczos vectors*. The $n \times n$ matrix Q from above has been replaced by the $n \times m$ matrix V_m and the $n \times n$ tridiagonal matrix T has been replaced by T_m which is also tridiagonal but is now $m \times m$. The Lanczos iteration's utility is only realized when $m \ll n$. We have an equivalent and more compact

version of (3.2) given by

$$AV_m = V_{m+1}\widetilde{T}_m \tag{3.3}$$

where

$$V_{m+1} = \begin{pmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_{m+1} \\ | & | & & | \end{pmatrix}$$

and

$$\widetilde{T}_m = \begin{pmatrix} \alpha_1 & \beta_1 & & \cdots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{m-1} \\ 0 & \cdots & & \beta_{m-1} & \alpha_m \\ 0 & \cdots & & 0 & \beta_m \end{pmatrix}$$

We can now supply the algorithm for the Lanczos Iteration as it is implemented in this thesis. This form of the iteration is taken from [13]:

The Lanczos Iteration

Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$ and a vector $b \in \mathbb{R}^n$ chosen by the user, the Lanczos iteration computes the matrices T_m and V_m from (3.2):

$\beta_0 = 0, \quad v_0 = 0, \quad v_1 = b/\|b\|$

for $i = 1, \ldots, m$

$w = Av_i$

$\alpha_i = v_i^T w$

$w = w - \beta_{i-1}v_{i-1} - \alpha_i v_i$

$\beta_i = \|w\|$

$v_{i+1} = w/\beta i$

end for loop

The majority of the work in the Lanczos iteration is in the multiplication Av_i

which involves $O(n^2)$ operations if A is dense, but for sparse matrices this will require significantly less work. More precisely, if A has about p nonzero entries per row on average, then it takes $(2p+8)mn$ flops to compute V_m and T_m, resulting in $O(n^2m)$ operations in the worst case [4],[13].

All of the matrices considered here come from finite difference discretizations, and so they are usually positive definite and sparse. Requiring higher order discretizations or Neumann boundary conditions could destroy positive definiteness. Matrices arising from spectral discretizations are also incompatible with this method. For example, if we were to use a Chebyshev differentiation matrix which is dense and not symmetric, our algorithm would be useless.

Another version of the iteration does not fix the loop duration as we have done here. Usually, the iteration runs until β_k becomes smaller than some user defined tolerance [4]. However, this could result in m being large and for our purposes the utility of the Lanczos iteration is greatest when $m \ll n$.

In general, there are two drawbacks to the Lanczos iteration that are worth mentioning. The first is that if larger values of m need to be taken, then the matrix V_m requires significant storage overhead if it is to be formed explicitly for future use. To remedy this, algorithms were designed to make more efficient use of the already computed Lanczos vectors. More precisely, after running the Lanczos iteration for m steps, we would choose a vector $\tilde{v} \in \text{span}\{v_1, \ldots, v_m\}$ and restart the Lanczos iteration with $\tilde{v}$ as the initial vector [4]. Since m never exceeds 30 in the present setting, we won't have the need for such restarted algorithms.

The other drawback is the loss of orthogonality among the Lanczos vectors. This problem can be rectified with algorithms employing so-called complete or selective orthogonalization techniques. Briefly, these algorithms orthogonalize the next Lanczos vector with respect to all or a subset of the previous Lanczos vectors, rather than orthogonalizing against only the previous two [4],[13]. But again, for the values of

m considered here, loss of orthogonality will not be an issue.

We don't provide the Arnoldi algorithm here because it is not used in this thesis, but it is worth mentioning the differences between it and the Lanczos iteration. The Arnoldi computes the partial *Hessenberg* reduction of a general matrix, and consequently a simple three term recurrence relation like (3.1) doesn't exist. Instead, the third line of the Lanczos iteration is replaced by an i-term recurrence relation which takes the form of another for loop. That Arnoldi is slower is apparent, and it too can suffer from large storage requirements. Restarted Arnoldi methods that use existing Arnoldi vectors were invariably devised. Loss of orthogonality can also be problematic with Arnoldi, and orthogonalization techniques have been implemented in versions of the iteration [4].

3.2 The Lanczos Approximation

Recall that every appearance of the matrix exponential e^{-kA} is accompanied only by its action on a vector $b \in \mathbb{R}^n$. This will allow us to avoid explicit computation of the matrix exponential if we utilize the Lanczos iteration. For a symmetric matrix A, the following observations will facilitate employing the Lanczos iteration in the computation of $e^{-kA}b$. Since the columns of V_m are orthogonal, $V_m^T V_m = I_m$ and $V_m^T v_{m+1} = 0$, and this yields

$$V_m^T A V_m = T_m. \tag{3.4}$$

In other words, T_m is the projection of A onto the mth Krylov subspace with respect to the basis $\{v_1, \ldots, v_m\}$, which is just the orthonormalized basis for $\mathcal{K}_m(A, b)$ [11]. The eigenvalues of T_m are known as the *Ritz values* of A.

The idea now is to seek an approximation of $e^{-kA}b$ that belongs to $\mathcal{K}_m$. Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$ and a vector $b \in \mathbb{R}^n$, suppose T_m and V_m are the results of the Lanczos iteration. Then $V_m V_m^T b$ is the orthogonal projection of b onto

$\mathcal{K}_m(A, b)$. With this in mind, Saad proposed the following approximation in [3],[11]:

$$e^{-A}b \approx \beta V_m e^{-T_m} e_1 \tag{3.5}$$

To justify this approximation, we need the following results also presented in [3].

Theorem 2.1 *Let A be an $n \times n$ symmetric matrix, let V_m and T_m be the results of m steps of the Lanczos iteration, and let $b \in \mathbb{R}^n$ have unit norm. Then for any polynomial p_j of degree $j \leq m-1$ we have*

$$p_j(A)b = V_m p_j(T_m) e_1 \tag{3.6}$$

We will also need what appears as a lemma in [3]:

Lemma 2.2 *Let A be any matrix whose minimal polynomial is of degree ν and f a function in the complex place which is analytic in an open set containing the spectrum of A. Moreover, let $p_{\nu-1}$ be the interpolating polynomial of the function f, in the Hermite sense, at the roots of the minimal polynomial of A, repeated according to their multiplicities. Then*

$$f(A) = p_{\nu-1}(A) \tag{3.7}$$

If the off-diagonal entries of T_m are nonzero, then the geometric multiplicity of each eigenvalue is 1, meaning the characteristic polynomial and the minimal polynomial are one and the same. The previous lemma then implies that

$$e^{T_m} = p_{m-1}(T_m) \tag{3.8}$$

where p_{m-1} interpolates the exponential function at the Ritz values in the Hermite sense, meaning that at a given point, p_{m-1} and the exponential function, along with their first $m-1$ derivatives, agree at that point [3].

Theorem 2.3 *The approximation* (3.5) *is equivalent to approximating* $e^A v$ *by* $p_{m-1}(A)v$ *where* p_{m-1} *is the unique polynomial of degree* $m-1$ *which interpolates the exponential function in the Hermite sense on the set of Ritz values repeated according to their multiplicities.*

Since Krylov subspaces are invariant under scaling, meaning $\mathcal{K}_m(A,b) = \mathcal{K}_m(kA,b)$, we have

$$e^{-kA}b \approx \beta V_m e^{-kT_m} e_1 \tag{3.9}$$

and this will be the foundation of our approach for improving efficiency in the calculation of (2.4-6). For this reason, we will call the right hand side of (3.9) the *Lanczos Approximation* to $e^{-kA}b$.

Implementation of the Lanczos Approximation

Once V_m and T_m have been computed, we still need to compute the matrix exponential e^{-kT_m}, but this is a relatively cheap computation since m is usually no larger than 30. At this point, Gallopoulos and Saad choose to compute the matrix exponential e^{-kT_m} using a rational Chebyshev approximation, instead of a Padé approximation. However, we will continue to use Padé approximation to compute e^{-kT_m}, specifically the diagonal $(6,6)$-Padé approximant, since this is usually within machine precision to what one obtains using `expm`. The algorithm that computes this $(6,6)$-Padé approximant is taken from [12] and has the same $O(m^3)$ complexity as Matlab's `expm` function, but it is slightly quicker.

To compute the right hand side of (3.9), we first need to compute T_m and V_m, then compute e^{-kT_m} and finally we need to make the necessary multiplications. This amounts to $O(n^2m) + O(m^3) + O(nm) \approx O(n^2m)$ operations when $m \ll n$. If we were to compute the matrix exponential on left hand side of (3.9) using the scaling and squaring method, it would take $O(n^3)$ operations and the computational savings is significant when n is very large.

The quality of the Lanczos approximation needs to be addressed. The following estimates were given in [11]. If the eigenvalues of the symmetric part of the matrix A are non-negative, then

$$\|e^{-kA}b - \beta V_m e^{-kT_m} e_1\| \leq 2\beta \frac{(k\rho)^m}{m!} \tag{3.10}$$

where ρ is the spectral radius of A,

$$\rho(A) = \max\{|\lambda| : \lambda \text{ is an eigenvalue of } A\}$$

and $\beta = \|b\|$. If in addition A is symmetric and positive definite, then

$$\|e^{-kA}b - \beta V_m e^{-kT_m} e_1\| \leq \beta \frac{(k\rho)^m}{2^{m-1}m!} \tag{3.11}$$

If A is positive definite and its eigenvalues are in the interval $[0, 4\gamma]$, Hochbruck and Lubich provide the sharper estimate in [5], which proves superlinear convergence for $m \geq 2\gamma k$:

$$\begin{aligned} &\|e^{-kA}b - \beta V_m e^{-kT_m} e_1\| \leq 10\beta e^{-m^2/(5\gamma k)}, && \sqrt{4\gamma k} \leq m \leq 2\gamma k \\ &\|e^{-kA}b - \beta V_m e^{-kT_m} e_1\| \leq 10\beta(\gamma k)^{-1} e^{-\gamma k} \left(\frac{e\gamma k}{m}\right)^m, && m \geq 2\gamma k \end{aligned}$$

For a fixed matrix A, it is apparent from these bounds that we can control the precision of the Lanczos approximation by choosing m large enough for a fixed time step k, or if we want m to remain small we can choose the time step k small enough.

Now that we have some justification of (3.9) and a priori error estimates, we can try to implement this approximation in the computation of (2.4-6). First we need to compute a_n, b_n and c_n. Note that there are two occurrences of $e^{-kA/2}$ in each of these, so we will try to consolidate the vectors on which this matrix exponential is

acting. This gives

$$\begin{aligned}
a_n &= e^{-kA/2}\left[u_n - A^{-1}F(u_n,t_n)\right] + A^{-1}F(u_n,t_n) \\
b_n &= e^{-kA/2}\left[u_n - A^{-1}F(a_n,t_n+k/2)\right] + A^{-1}F(a_n,t_n+k/2) \\
c_n &= e^{-kA/2}\left[a_n - A^{-1}\left(2F(b_n,t_n+k/2) - F(u_n,t_n)\right)\right] \\
&\quad + A^{-1}\left(2F(b_n,t_n+k/2) - F(u_n,t_n)\right)
\end{aligned} \tag{3.12}$$

Since we won't compute the inverse A^{-1} explicitly we first write these as

$$\begin{aligned}
Aa_n &= e^{-kA/2}\left[Au_n - F(u_n,t_n)\right] + F(u_n,t_n) \\
Ab_n &= e^{-kA/2}\left[Au_n - F(a_n,t_n+k/2)\right] + F(a_n,t_n+k/2) \\
Ac_n &= e^{-kA/2}\left[Aa_n - \left(2F(b_n,t_n+k/2) - F(u_n,t_n)\right)\right] \\
&\quad + 2F(b_n,t_n+k/2) - F(u_n,t_n)
\end{aligned} \tag{3.13}$$

Before the time loop begins, we compute the Cholesky decomposition $A = LL^T$ in the case where A is positive definite, otherwise an LU decomposition can be used. Since all the matrices dealt with here are symmetric and positive definite the Cholesky is always used. To compute a_n, we use the approximation

$$e^{-kA/2}\left[Au_n - F(u_n,t_n)\right] \approx \|Au_n - F(u_n,t_n)\| V_m e^{-kT_m/2} e_1 \tag{3.14}$$

and compute the right side of (3.14), which is done with the author's function `expml` (see Appendix). Considering the approximation in (3.9), the function `expml` accepts inputs A, b, m, and k; the output is $\beta V_m e^{-kT_m} e_1$. The nonlinear term $F(u_n,t_n)$ is then added to (3.14) in an overwrite. Finally, a_n is obtained through two Matlab backslash solves with L and its transpose. The same process is carried out to obtain b_n and c_n. Finally, u_{n+1} is obtained in the same way as it is in [14], so the only difference in the way we compute ETDRK4 is in how a_n, b_n and c_n are obtained.

Here is the algorithm we call the *Lanczos method* of computing a_n, b_n, c_n, and u_{n+1} in the ETDRK4 formulae:

The Lanczos ETDRK4 Algorithm

First compute the Cholesky factorization of A so that $A = LL^T$, and choose the desired Krylov subspace dimension m.

1. To compute a_n, compute the product

$$z = e^{-kA/2}\left[Au_n - F(u_n, t_n)\right],$$

overwrite z:

$$z = z + F(u_n, t_n)$$

and solve the systems

$$Ly = z \qquad \text{and} \qquad L^T a_n = y$$

2. To compute b_n, compute the product

$$z = e^{-kA/2}\left[Au_n - F(a_n, t_n + k/2)\right],$$

overwrite z:

$$z = z + F(a_n, t_n + k/2)$$

and solve the systems

$$Ly = z \qquad \text{and} \qquad L^T b_n = y$$

3. Similarly, to compute c_n, compute the product

$$z = e^{-kA/2}\left[Aa_n - (2F(b_n, t_n + k/2) - F(u_n, t_n))\right],$$

overwrite z:

$$z = z + 2F(b_n, t_n + k/2) - F(u_n, t_n)$$

and solve the systems

$$Ly = z \qquad \text{and} \qquad L^T c_n = y$$

4. Finally, to compute u_{n+1}, first solve

$$\begin{aligned}(kA - c_1 I)Nu_1 = {} & w_1 u_n + k w_{11} F(u_n, t_n) \\ & + k w_{21}\Big(F(a_n, t_n + k/2) + F(b_n, t_n + k/2)\Big) \\ & + k w_{31} F(c_n, t_n + k),\end{aligned}$$

for Nu_1 and then compute

$$u_{n+1} = u_n + 2\,\Re\left(Nu_1\right).$$

It is apparent now that u_{n+1} is obtained in the same way as in the $(2,2)$-Padé algorithm, and that the only difference is in how a_n, b_n, and c_n are computed. For a more thorough comparison, suppose A is a symmetric $n \times n$ matrix from a spatial discretization and that $m \ll n$ is the chosen Krylov subspace dimension. Then the computation of the $(2,2)$-Padé method requires solving four $n \times n$ linear systems where the coefficient matrices $(kA - c_1 I)$, $(kA - \widetilde{c}_1 I)$ and the known vectors are all complex valued. Compare this to the Lanczos ETDRK4 algorithm which must

compute m real valued $n \times n$ matrix-vector multiplications, and then solve two real $n \times n$ triangular systems.

In the case where A is the tridiagonal matrix coming from the one dimensional central difference discretization, we will find that the $(2, 2)$-Padé algorithm is faster. But when A has a more complex structure, as in the case of a two dimensional central difference discretization, the Lanczos method will be faster. In any case, A must be symmetric, positive definite, and sparse. If it is not symmetric, then the Arnoldi iteration can be used in place of the Lanczos iteration. If it is not positive definite, then the Cholesky factorization must be substituted with a sparse LU decomposition, for example. However, sparsity cannot be sacrificed as both iterations are ineffective on dense matrices.

Chapter 4

Numerical Experiments

In this chapter numerical experiments are conducted to support the use of Krylov subspaces in computing ETDRK4. Before we see how the Lanczos approximation affects the computation of some PDE, it will be beneficial to see how it performs at a lower level – this is the topic of the first section. In the next section we consider the one dimensional Allen-Cahn equation and some methods of solving it numerically. In the final section the two dimensional Allen-Cahn equation will be the subject of our numerical investigation.

4.1 Quality of the Lanczos Approximation

We will first demonstrate the quality of the Lanczos approximation for a few different matrices. All of the matrices we will encounter in this thesis are symmetric and positive definite. To see how the Krylov subspace dimension affects the Lanczos approximation, we will investigate the behavior of $\|e^{-kA}b - \beta V_m e^{-kT_m} e_1\|$ with respect to m. Recall that for symmetric matrices, we have $\|A\| = \rho(A)$, so the bounds provided in [11] and [5] depend on the norm of A, m, and k.

For all of the following examples, we compute the "true" solution with Matlab's `expm` function and we compute the Lanczos approximation with the author's function `expml`, which uses the Lanczos iteration to compute V_m and T_m, and uses the scaling and squaring based $(6, 6)$-Padé approximant from [12] to compute e^{-kT_m}.

The first matrix to consider is the second order central difference matrix assuming

homogeneous Dirichlet boundary conditions

$$A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & -1 & 2 & \ddots & \\ \vdots & & \ddots & \ddots & -1 \\ 0 & \cdots & & -1 & 2 \end{pmatrix}$$

This matrix is positive definite, symmetric, and tridiagonal. Let A be 1024×1024, let v be the normalized vector of all ones (so $\beta = 1$), let $k = 0.1$, and let $h = 1$. It turns out that $\rho(A) = \|A\| = 4$. In Figure 4.1 we plot the logarithm of the actual error $\|e^{-kA}v - V_m e^{-kT_m}e_1\|$ and the theoretical bounds from [5] and [11] against the Krylov subspace dimension m. It is apparent that the error committed follows the theoretical bounds until the actual error reaches approximately machine precision at about 10 iterations of the Lanczos iteration. Decreasing the time step k causes the actual error to reach machine precision at progressively fewer steps of the Lanczos iteration, and increasing it has the opposite effect. For example, setting $k = 0.01$ results in the actual error reaching machine precision in 5 iterations.

The second matrix we investigate is the fourth order central difference matrix assuming homogeneous Dirichlet boundary conditions. This matrix is again positive definite, but now it is pentadiagonal:

$$A = \frac{1}{12h^2}\begin{pmatrix} 30 & -16 & 1 & 0 & \cdots & 0 \\ -16 & 30 & -16 & 1 & & \vdots \\ 1 & -16 & 30 & -16 & \ddots & \\ 0 & 1 & -16 & 30 & \ddots & 1 \\ \vdots & & \ddots & \ddots & \ddots & -16 \\ 0 & \cdots & & 1 & -16 & 30 \end{pmatrix}$$

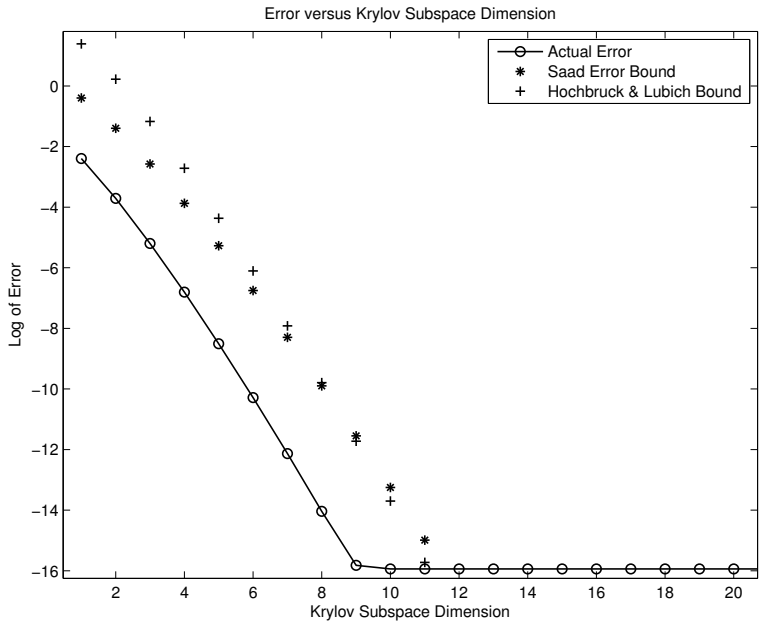


Figure 4.1: Theoretical and Actual Error versus Krylov Subspace Dimension for Tridiagonal A

Assume again that A is 1024×1024, let v be as before and let $k = 0.1$, $h = 1$. In this case A has a larger norm: $\rho(A) = \|A\| = 64$, and we expect the convergence to be slower. This is indeed the case, by Figure 4.2, since the actual error doesn't reach its minimum (which isn't quite at machine precision) until 20 Lanczos iterations. That the bound provided in [5] is sharper is also apparent here. Setting $k = 0.01$ has the same effect as before, and in this case the actual error almost reaches machine precision in 10 iterations.

Next we consider a 1024×1024 banded positive definite matrix A with bandwidth 41 and band density equal to 1, i.e. all entries in the band are nonzero. This matrix has norm $\|A\| \approx 34$ and v and k are as before, so we expect the speed of convergence to be between the previous two. As Figure 4.3 shows, the actual error has decreased to approximately machine precision by 16 Lanczos iterations. Once again, taking $k = 0.01$ results in the error converging to approximately machine precision in 9 iterations. These experiments verify that the Lanczos approximation adheres to the

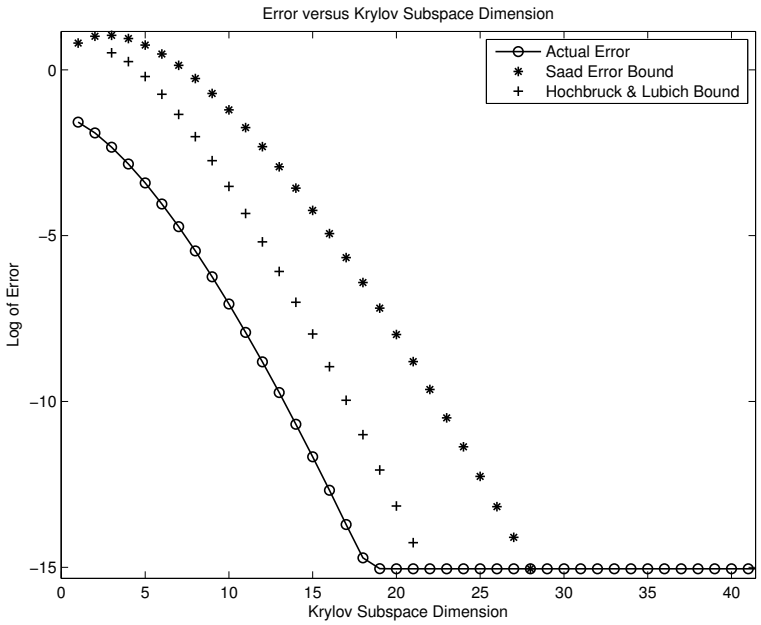


Figure 4.2: Theoretical and Actual Error versus Krylov Subspace Dimension for Pentadiagonal A

bounds in [5] and [11], which for a fixed time step k, is determined by the norm of the matrix.

If we scaled the previous matrices so that they had equal norms, then the previous three figures would be almost indistinguishable because the speed of convergence depends solely on the norm of the matrix, for fixed k. In other words, the structure of the matrix (provided it is symmetric) has no effect on the accuracy of the Lanczos approximation. However, the structure of the matrix does affect the time it takes to compute this approximation. We stated before that the function `expml` that computes this approximation has complexity $O(n^2m) + O(m^3) + O(nm) \approx O(n^2m)$ if $m \ll n$. Figure 4.4 displays how the CPU time of `expml` is affected by increasing m for different matrices. To isolate the effect the structure of the matrix has on CPU time, we scaled all matrices to have a norm of 4 and set $k = 0.1$ and $n = 1024$. Observe that in the tridiagonal and pentadiagonal cases the CPU time grows linearly with m. However, we notice higher order growth with increasing m in the case where

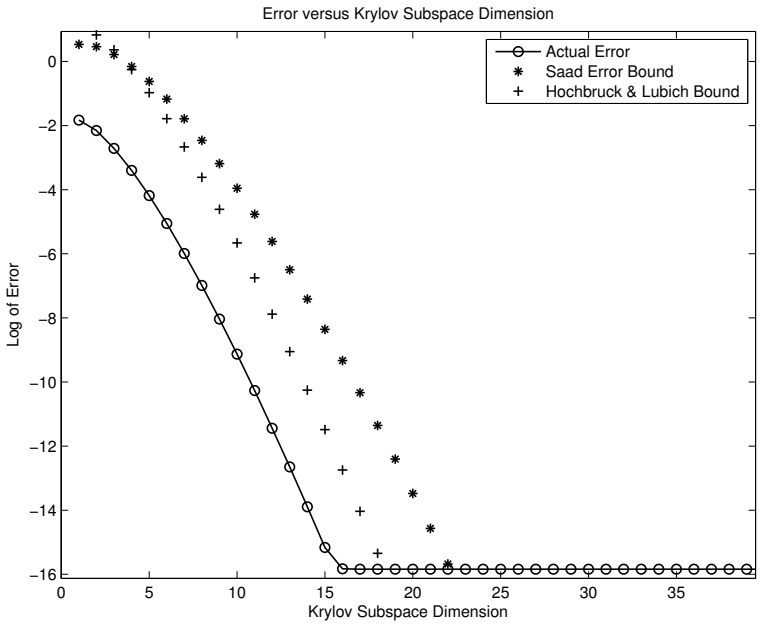


Figure 4.3: Theoretical and Actual Error versus Krylov Subspace Dimension for A with bandwidth 41

A has larger bandwidth. The linear growth in m when A has bandwidth 3 or 5 leads us to conclude that the algorithm for computing the $(6,6)$-Padé approximant has complexity closer to $O(m)$ for input matrices of small bandwidth. Moreover, the cubic growth in m doesn't take noticeable effect until the matrix is reasonably dense, as in the case of the matrix with bandwidth 41.

4.2 One dimensional Allen-Cahn Equation

In this chapter we will apply the new method of computing ETDRK4 that incorporates the Lanczos iteration to a PDE that gives rise to a stiff system of ODE's upon discretization. The PDE subject to the experiment is the one dimensional Allen-Cahn equation. It is a nonlinear reaction diffusion equation with stable equilibria at ± 1 and an unstable equilibrium at 0. Many different physical processes are modeled by the Allen-Cahn, most notably it is able to describe how the boundaries between phases of iron alloys change over time [7]. Let $u = u(x,t)$ be the concentration of

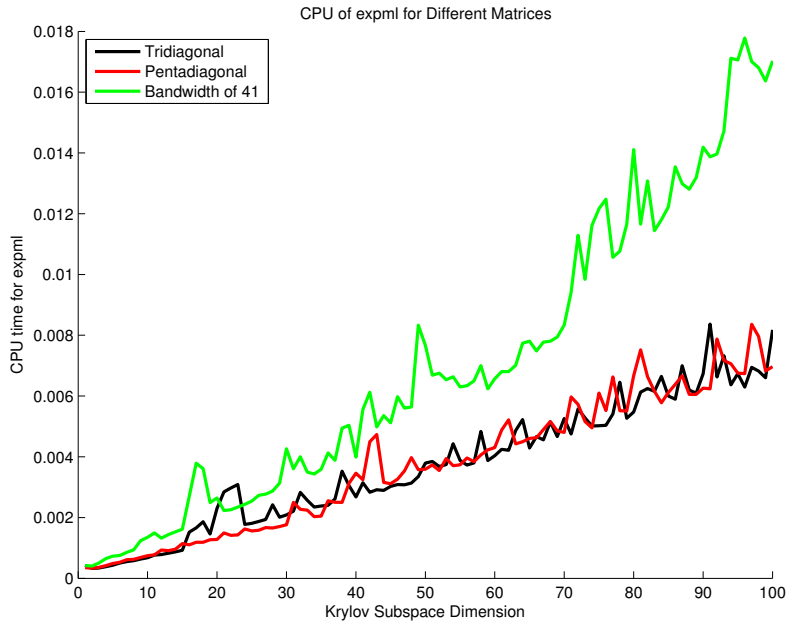


Figure 4.4: CPU time of expml function for Different matrices versus Krylov Subspace Dimension m

an alloy at the point x at time t. If $u = 1$, then only one alloy is present; if $u = -1$, then only the other alloy is present. Given some boundary conditions on a domain Ω (here $\Omega = [-1, 1]$) and an initial concentration $u_0(x) = u(x, 0)$, the Allen-Cahn equation is given by

$$u_t = \varepsilon u_{xx} + F(u) \tag{4.1}$$

with $x \in [-1, 1]$, $t \geq 0$, and $F(u) = u - u^3$. The boundary conditions are $u(-1, t) = -1$ and $u(1, t) = 1$ for all $t > 0$, and we take the initial data

$$u_0(x) = 0.53x + 0.47 \sin(-1.5\pi x)$$

as in [7]. The number ε is a parameter which we fix at $\varepsilon = 0.01$.

Choose a uniform discretization of $[-1, 1]$ consisting of the M points $x_1, x_2, \ldots, x_M$ and let u_j denote the approximation of $u(x_j)$ at x_j for $j = 1, \ldots, M$. Approximate

u_{xx} by the three point stencil

$$\frac{u_{j-1} - 2u_j + u_{j+1}}{h^2} \tag{4.2}$$

By the boundary conditions, we only need to solve for $u_2, u_3, \ldots, u_{M-1}$ at each time step. The stiff linear part of (4.1), u_{xx}, is treated implicitly, while the nonlinear term $F(u)$ is treated explicitly. The matrix A is the negative definite version from the first example in Section 4.1:

$$A = \begin{pmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & & \vdots \\ 0 & 1 & -2 & \ddots & \\ \vdots & & \ddots & \ddots & 1 \\ 0 & \cdots & & 1 & -2 \end{pmatrix}$$

This is a second order in space discretization and we use this, as opposed to a fourth order discretization, because the resulting matrix is symmetric. If a fourth order discretization was used, one sided finite differences at the boundary points would be necessary to maintain the order and this destroys the symmetry of the matrix. Orthogonalized Krylov subspaces for non symmetric matrices are produced with the Arnoldi method, which is slower than the Lanczos iteration. Designing an algorithm to compute the approximation in (3.9) for non symmetric matrices would not be difficult – it just requires using the Arnoldi method instead of Lanczos. The solution of (4.1) is plotted in Figure 4.5 for $k = 1$, $t = 70$, and $M = 50$ domain points – the $(2, 2)$-Padé ETDRK4 scheme was used to compute this.

In one dimension, the $(2, 2)$-Padé scheme is 10-15 times faster than than the Lanczos method, assuming a Krylov subspace dimension of 15. This discrepancy is a result of how the vectors a_n, b_n, and c_n are computed. With the $(2, 2)$-Padé scheme, finding a_n, for example, requires one solve of an $M - 2 \times M - 2$ tridiagonal

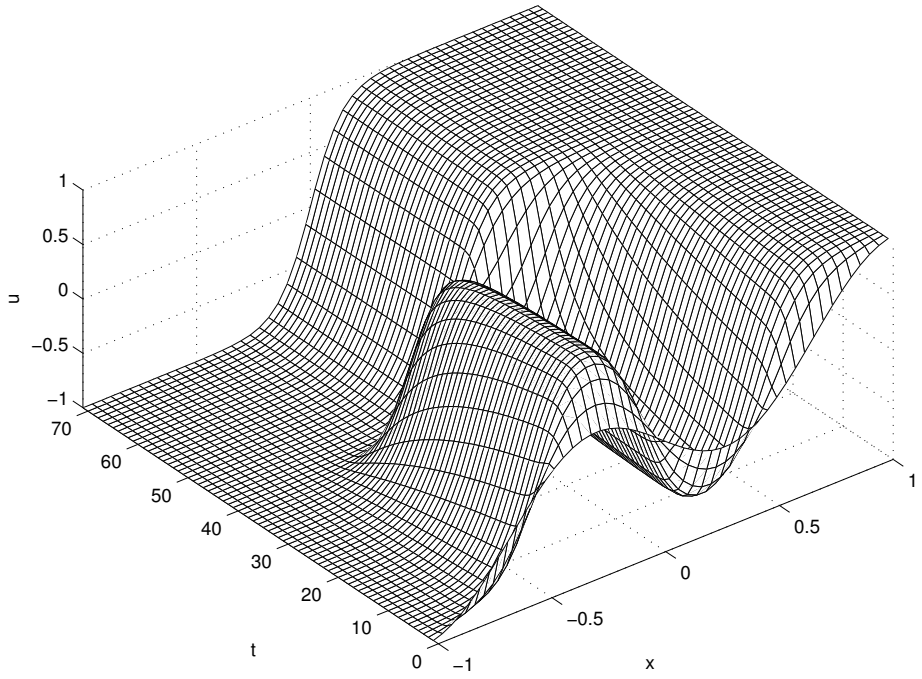


Figure 4.5: Solution of 1-D Allen-Cahn Equation with $k = 1$, $t = 70$, $M = 50$

linear system where the coefficient matrix $kA - \tilde{c}_1 I$ and the known vector contain complex numbers, coming from poles of the Padé approximant. This is done with Matlab's backslash solver, which employs LAPACK's banded solver.

When a_n is computed with the Lanczos method, there are m real valued matrix-vector multiplications (the Lanczos iteration), the computation of the $(6, 6)$-Padé approximation of $e^{-kT_m/2}$, the multiplication of V_m with the first column of $\widetilde{R}_{6,6}(kT_m)$, and finally two triangular solves involving the Cholesky factor of A. Both triangular solves are done in half the time it takes to solve the system $(kA - \tilde{c}_1 I)x = b$ from above. It is clear then that the primary hindrance to speed in the Lanczos method is the Lanczos iteration itself.

Figure 4.6 shows that the relative errors computed at $t = 3$ seem to coincide, but upon closer inspection one can see the errors are in fact unequal. These figures support the fact that the Lanczos method is merely another way to compute a_n, b_n

and c_n, possibly even u_{n+1}. The errors are the same, but they differ in time. In this one dimensional example the Padé method is faster, but in two dimensions the Lanczos method will be faster.

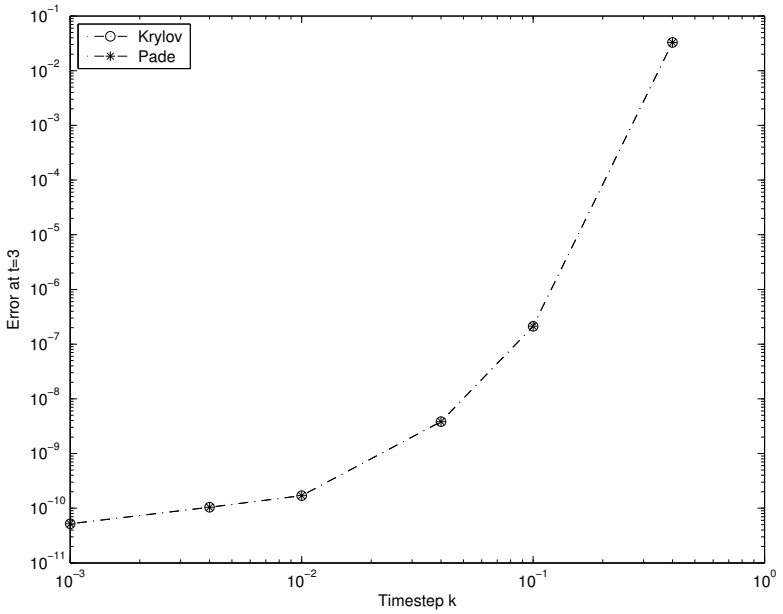


Figure 4.6: 1D Allen-Cahn relative errors versus time step with $M = 50$

It should be noted that when the time step k is small enough, we can set m to be accordingly small so that less matrix-vector multiplications have to be computed. The reason for this is apparent from the theoretical error bound in [5] and [11]. This motivates the idea of adapting m and k for each time step. Efficiency can be greatly improved if k and m are chosen adaptively [9].

4.3 Two dimensional Allen-Cahn Equation

In two dimensions the domain becomes $\Omega = [-1, 1] \times [-1, 1]$ and (4.1) becomes

$$u_t = \varepsilon \Delta u + F(u) \tag{4.3}$$

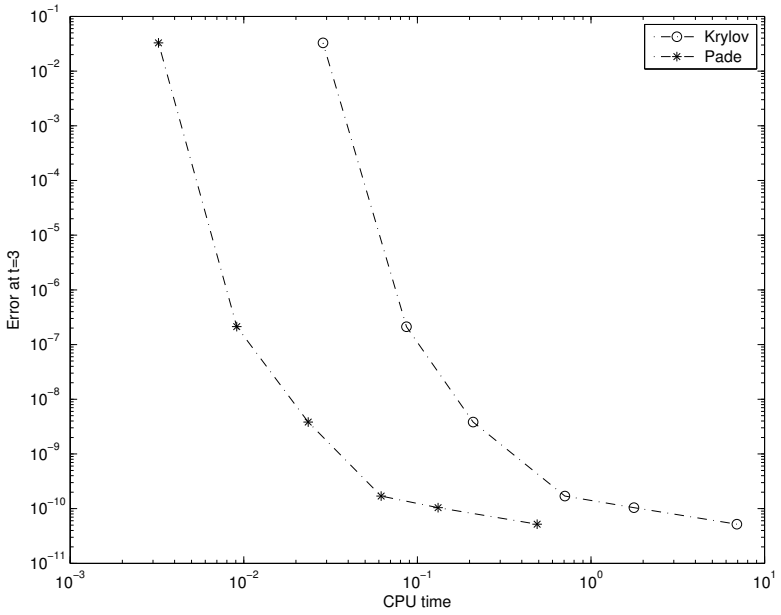


Figure 4.7: 1D Allen-Cahn relative errors versus CPU time with $M = 50$

where $\Delta u = u_{xx} + u_{yy}$ and $F(u)$ is as before. A common choice for an initial concentration is a function exhibiting some randomness, simulating a heterogeneous initial mixture of two alloys, so $u \approx 0$ would be an appropriate choice. The boundary conditions accompanying this initial data are usually homogenous Neumann on all of $\partial\Omega$. Discretize Ω with the M^2 uniformly spaced points (x_i, y_j), $i, j = 1, \ldots, M$, and let u_{ij} denote the approximation of $u(x_i, y_j)$ at the point (x_i, y_j). Approximate the Laplacian by the 5 point stencil

$$\frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}}{h^2} \tag{4.4}$$

where $h = |x_i - x_{i+1}| = |y_i - y_{i+1}|$. If we take the standard row ordering of the solutions and assume homogeneous Neumann boundary conditions, the coefficient matrix obtained will be singular. This is not a problem for the $(2, 2)$-Padé scheme because there are no linear systems solved whose coefficient matrix is A. The Lanczos method of computing the solution does, however, encounter a problem when

solving the linear systems in equation (3.13) because here the coefficient matrix is A. For this reason, inhomogeneous Dirichlet boundary conditions will be imposed with $u = 1$ on $\partial\Omega$. This is a legitimate assumption considering the corresponding physical interpretation: there is only one alloy present at the boundary.

The resulting coefficient matrix A is the $(M-2)^2 \times (M-2)^2$ symmetric, negative definite, block tridiagonal matrix:

$$A = \begin{pmatrix} B & I & O & \cdots & O \\ I & B & I & & \vdots \\ O & I & B & \ddots & \\ \vdots & & \ddots & \ddots & I \\ O & \cdots & & I & B \end{pmatrix}$$

where

$$B = \begin{pmatrix} -4 & 1 & 0 & \cdots & 0 \\ 1 & -4 & 1 & & \vdots \\ 0 & 1 & -4 & \ddots & \\ \vdots & & \ddots & \ddots & 1 \\ 0 & \cdots & & 1 & -4 \end{pmatrix}$$

is $M-2 \times M-2$, I is the $M-2 \times M-2$ identity matrix, and O is the $M-2 \times M-2$ matrix of zeros.

To compare the $(2,2)$-Padé algorithm and the Lanczos method we take the following setup of the 2D Allen-Cahn equation. Assume homogeneous Dirichlet boundary conditions with $u = 1$ on $\partial\Omega$, let $M = 50$, and assume an initial concentration given by a sum of three slightly different Gaussians at some distance apart, see Figure 4.10. We will integrate up to $T_{max} = 5$ and compare errors there. The matrix A has norm $\|A\| \approx 50$, so for the Lanczos method we take $m = 25$ for time steps of $k = 0.001, 0.01, 0.1$. For $k = 0.001$ and $k = 0.01$ this is may be an overly cautious

choice for m, since the error committed by the Lanczos approximation is at machine precision by $m = 10$ for these values of k. For $k = 0.1$, though, this choice for m seems reasonable.

Figure 4.8 demonstrates that the Lanczos Method is merely another way of computing ETDRK4 using the $(2, 2)$-Padé algorithm since the errors coincide for the time steps used. Figure 4.9 shows that the Lanczos method is faster than the $(2, 2)$-Padé method. More precisely, the Lanczos method is 1.9 times faster and if we had chosen m more judiciously this factor would be greater.

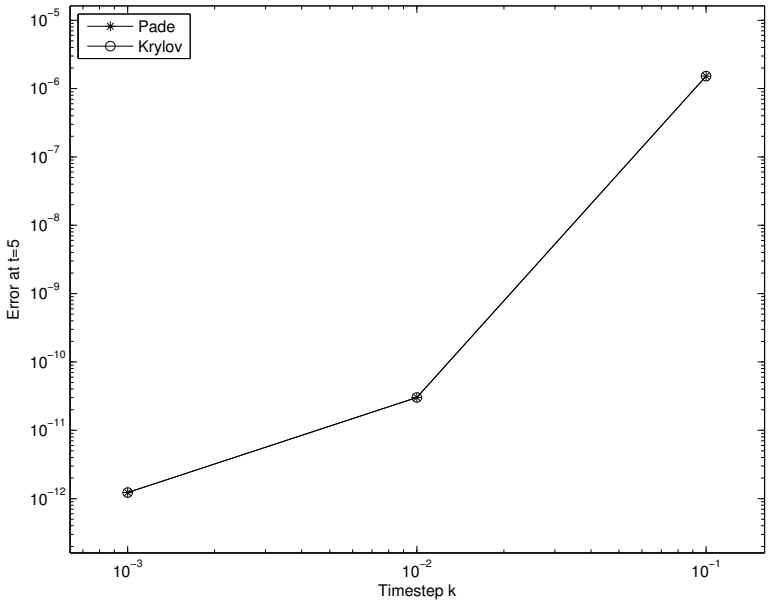


Figure 4.8: 2D Allen-Cahn relative errors versus time step with $M = 50$

To see why the Lanczos method is faster, first recall that the $(2, 2)$-Padé algorithm consists of four $(M-2)^2 \times (M-2)^2$ complex block tridiagonal linear systems that need to be solved for each time step. Compare this to the Lanczos method where the computationally intensive operations include m real matrix-vector multiplications and two real, sparse triangular solves. The values of m experimented with here are all less than 30. The m matrix-vector multiplications eclipse the two triangular solves in terms of CPU time so we will focus on the multiplications. With

respect to CPU time, the disparity between solving one complex block tridiagonal linear system and performing m real block tridiagonal matrix-vector multiplications is explained by the structure of the matrix. It simply takes longer to solve a block tridiagonal system than it does to perform m matrix-vector multiplications. This disparity was absent in the one dimensional case because the difference in CPU time to solve a *tridiagonal* system and to make a matrix-vector multiplication is relatively small. Therefore, performing m matrix-vector multiplications will be slower than one solve if the matrix is tridiagonal. We can infer that if we have a discretization producing a symmetric, positive definite matrix with a more complex sparsity pattern, then the difference in CPU time will be greater.

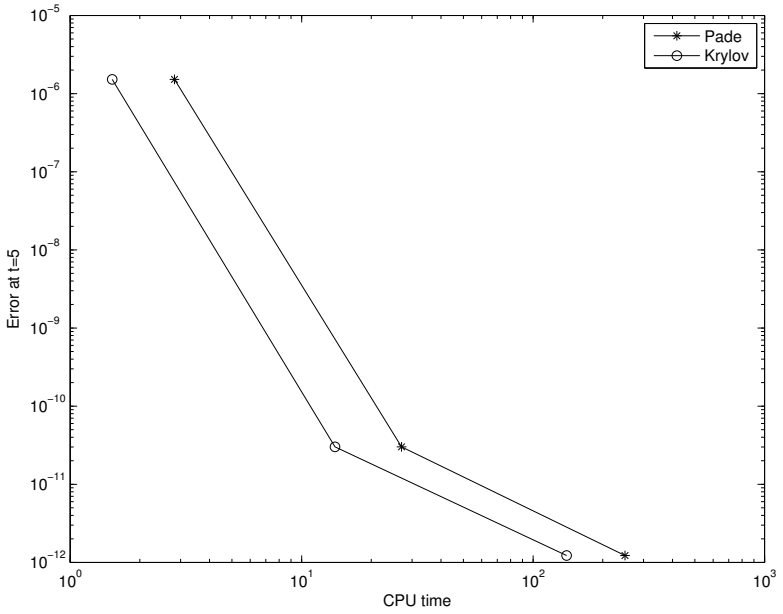
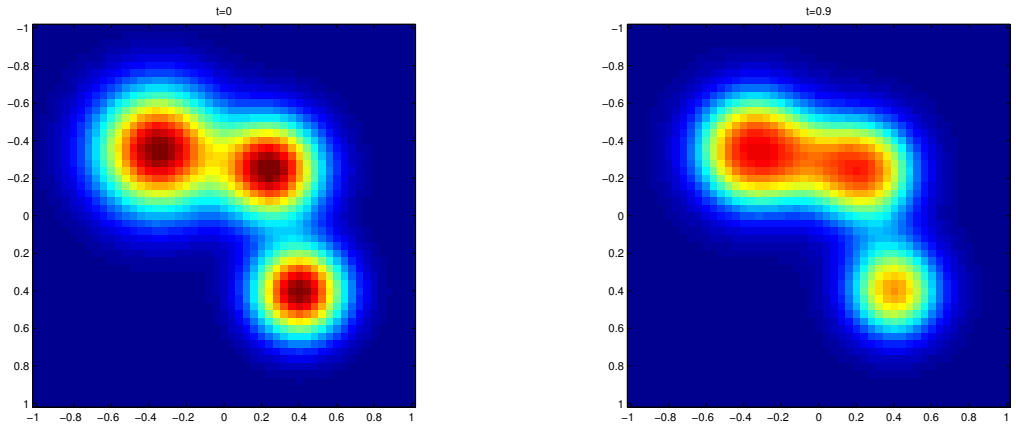


Figure 4.9: 2D Allen-Cahn relative errors versus CPU time with $M = 50$

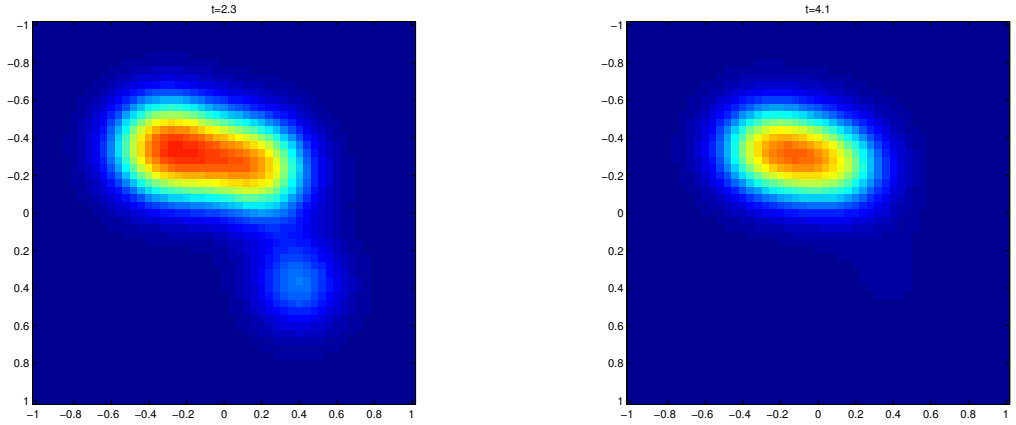
Figures 4.10 and 4.11 illustrate the time evolution of the 2D Allen-Cahn equation. The blue represents a concentration of $u = 1$, so the initial condition here is that there are three somewhat isolated concentrations of only the other alloy, i.e. we have $u = -1$ at the center of each of these Gaussian concentrations. The behavior of the solution through time is that the isolated areas of concentration with $u = -1$



(a) $t = 0$ (b) $t = 0.9$

Figure 4.10: Solution to 2D Allen-Cahn equation with $M = 50$

tend toward the stable equilibrium $u = 1$, and at $t \approx 6$ only one concentration remains. When an even initial mixture of the alloys is assumed ($u_0(x, y) \approx 0$) and homogeneous Neumann boundary conditions are imposed, the behavior is not so predictable.



(a) $t = 2.3$ (b) $t = 4.1$

Figure 4.11: Solution to 2D Allen-Cahn equation with $M = 50$

Chapter 5

Conclusions

An algorithm that implements the ETDRK4 scheme, while incorporating Krylov subspace methods, has been developed. Here, we focused on employing the Lanczos iteration in the computation of the formulae appearing in ETDRK4 since all the matrices considered here were symmetric. Using Saad's approximation we were able to approximate the action of the matrix exponential on a vector and apply this result to calculation of the terms in ETDRK4. In one dimension, we found that the Padé algorithm is faster at approximating solutions to the Allen-Cahn equation because the algorithm that solves a tridiagonal system has $O(n)$ complexity. In two dimensions we found that the Lanczos algorithm is about twice as fast as the Padé algorithm at computing these solutions due to a more complex sparsity pattern in the matrix. In summary, incorporating the Lanczos iteration into the Padé scheme improved efficiency in calculating solutions to the two dimensional Allen-Cahn equation assuming a spatial discretization that produces a symmetric matrix is used.

The choices of m and k here were not chosen as wisely as possible. Ideally, we would have an algorithm that steps through time and adapts m and k efficiently, so that no extra Krylov subspace are used. The authors of [9] devise an adaptive algorithm like this, and this should be referenced for future development as this system could significantly improve efficiency.

We would also like to use fourth order spatial discretizations, as this would increase accuracy. When this is implemented, it might very well be the case that symmetry is lost in the matrix. The Arnoldi method would now be required to

orthonormalize the Krylov subspaces. There may also be room for further improvement if we utilize the so – called φ-functions in applying the Lanczos or Arnoldi approximation. This route is further discussed in [6],[9], and [10]. Preconditioning the matrix exponential in order to achieve faster convergence is another promising direction [2].

Other discretizations producing sparse matrices should be experimented with. For example, finite element methods can produce sparse, symmetric, positive definite matrices that have a more complex structure than that of banded or block-banded matrices. In this case, Krylov subspace methods are very promising in terms of CPU time. More efficient direct solvers should also be considered when the matrix at hand comes from the two dimensional discretized Laplacian. Cyclic reduction and multigrid methods are two possible options. Investigating how Krylov subspace methods and their implementation in ETDRK4 (and possibly other exponential integrators) performs on other reaction diffusion equations is another future prospect.

Bibliography

[1] S.M. Cox and P.C. Matthews, *Exponential Time Differencing for Stiff Systems,* J. Comp. Phys.**176** (2002) 430-455.

[2] J. van den Eshof and M. Hochbruck, *Preconditioning Lanczos Approximations to the Matrix Exponential,* SIAM J. Sci. Comput. **27(4)** (2006) 1438-1457.

[3] E. Gallopoulos and Y. Saad, *Efficient Solution of Parabolic Equations by Krylov Approximation Methods,* SIAM J. on Sci. and Stat. Comput. **13(5)** (1992) 1236-1264.

[4] G. Golub and C. Van Loan, *Matrix Computations,* 3rd ed. (The Johns Hopkins University Press, Baltimore, 1996).

[5] M. Hochbruck and C. Lubich, *On Krylov Subspace Approximations to the Matrix Exponential Operator,* SIAM J. Num. Anal. **34(5)** (1997) 1911-1925.

[6] M. Hochbruck and A. Ostermann, *Exponential Integrators,* Acta Numerica **19** (2010) 209-286.

[7] A.K. Kassam and L.N. Trefethen, *Fourth-order Time Stepping for Stiff PDEs,* SIAM J. Sci. Comput. **26(4)** (2005) 1214-1233.

[8] C. Moler and C. Van Loan, *Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later,* SIAM Review **45** (2003) 3-49.

[9] J. Niesen, W. M. Wright, *A Krylov Subspace Algorithm for Evaluating the ϕ-functions Appearing in Exponential Integrators,* ACM Trans. Math. Software **38(3)** (2012) Article 22.

[10] M. Popolizio and V. Simoncini, *Acceleration Techniques for Approximating the Matrix Exponential Operator,* SIAM J. Matrix Anal. and Appl. **30(2)** (2008) 657-683.

[11] Y. Saad, *Analysis of Some Krylov Subspace Approximations to the Matrix Exponential Operator,* SIAM J. Num. Anal. **29** (1992) 209-228.

[12] R.B. Sidje, *Expokit: A Software Package for Computing the Matrix Exponentials,* ACM Trans. Math. Software **24(1)** (1998) 130-156.

[13] L.N. Trefethen and D. Bau, III, *Numerical Linear Algebra,* (SIAM, Philadelphia, 1997).

[14] B.A. Wade, A.Q.M. Khaliq, J. Martin-Vaquero, M. Yousef, *Smoothing Schemes for Reaction-Diffusion Systems with Nonsmooth Data,* J. of Comput. and Appl. Math. **223(1)** (2009) 374-386.

Appendix

The Lanczos Approximation Algorithm

```
function w = expml(A,b,m,k)
%
% Author: Jeffrey Allen
%
% The function expml computes an approximation to the product
% e(kA)*b using the partial tridiagonalization of A which uses the
% Lanczos iteration. The inputs are the SYMMETRIC matrix A,
% the vector b being acted on, the Krylov subspace dimension m,
% and the timestep k. The output is the approximation to e(kA)*b,
% due to Saad. The function padm() is from [12]

% make A sparse
A = sparse(A);

n = length(b);

% preallocate T,Q
T = zeros(m);
Q = zeros(n,m);

% normalize b
v1 = b/norm(b);

% Lanczos iteration j = 1
v = A*v1;
alpha = v1'*v;
v = v - v1*alpha;
Q(:,1) = v1;
T(1,1) = alpha;

% Lanczos iteration j = 2,...,m
for j = 2:m,

    beta = norm(v);
    v0 = v1;
    v1 = v/beta;

    v = A*v1 - v0*beta;
```

```
    alpha = v1'*v;
    v = v - v1*alpha;

    T(j,j-1) = beta;
    T(j-1,j) = beta;
    T(j,j) = alpha;
    Q(:,j) = v1;

end

T = sparse(T);

% computes (6,6)-Pade approximant to matrix exponential exp(kT)
E = padm(k*T);

% Lanczos approximation to the product exp(kA)*b
w = norm(b)*Q*E(:,1);

end
```

The (2,2)-Padé ETDRK4 Algorithm

```
%
% Author: Jeffrey Allen
%
% solves the 2D Allen Cahn equation via ETDRK4 time stepping and
% second order centered differences space discretization
%
% u_t = epsilon*(u_xx + u_yy) + F(u) on [-1,1]x[-1,1],
%
% where F(u) = u-u^3, and Dirichlet BC's with u=1
%
% ETDRK4 coefficients are computed using (2,2)-Pade scheme
%
% Script requires the functions: laplacian(), expml(), lanczos2(),
% and padm()

% poles, weights for r=s=2
c1 = -3.0 + 1.73205080756887729352*1i;
w1 = -6.0 - 10.392304845413263761 1*1i;
w11 = -0.5 - 1.4433756729740644112 7*1i;
w21 = -1.154700538379251529 01*1i;
w31 = 0.5 + 0.288675134594812882 25*1i;
tdc1 = -6.0 + 3.464101615137754587 0548*1i;
```

```
tdw1 = -12.0 - 20.78460969082652752232935*1i;
omega1 = -3.464101615137754587O5*1i;

% parameter
epsilon = 0.01;

% number of grid points per dimension
M = 50;

% 1D meshes
x = linspace(-1,1,M)';
y = x;

% 2D mesh
[X,Y] = meshgrid(x,y);

% spatial step
dx = 2/(M-1);

% initial data
U = 1 - 2*exp(-10*((X+.35).^2 + (Y+.35).^2)) - ...
        2*exp(-18*((X-.40).^2 + (Y-.40).^2)) - ...
        2*exp(-15*((X-.25).^2 + (Y+.25).^2));

% constructs discretized 2D Laplacian matrix
B = {'DD','DD'};
[~,~,A] = laplacian([M-2 M-2],B);
A = (epsilon/(dx^2))*A;
I = speye(size(A));

% time step
k = 0.1;

% maximum iterations
Tmax = 5; Nmax = round(Tmax/k);

% homogenize
V = U - 1;

% (M-2)^2 unknowns
V0 = V(2:M-1,2:M-1);

% solution matrix
W = zeros(M,M,Nmax);
```

```
% impose IC and BC
W(:,:,1) = padarray(V0 + 1,[1 1],1);

% column ordering
V0_vec = V0(:);

% LHS matrices for (2,2)-Pade
M1 = k*A - c1*I;
M2 = k*A - tdc1*I;

% time stepping
for i = 1:Nmax

    Fv = (V0_vec+1) - (V0_vec+1).^3;
    Na = M2\(tdw1*V0_vec + k*omega1*Fv);
    a = V0_vec + 2*real(Na);

    Fa = (a+1) - (a+1).^3;
    Nb = M2\(tdw1*V0_vec + k*omega1*Fa);
    b = V0_vec + 2*real(Nb);

    Fb = (b+1) - (b+1).^3;
    Nc = M2\(tdw1*a + k*omega1*(2*Fb - Fv));
    c = a + 2*real(Nc);

    Fc = (c+1) - (c+1).^3;
    Nu = M1\(w1*V0_vec + k*w11*Fv + k*w21*(Fa + Fb) + k*w31*Fc);
    V0_vec = V0_vec + 2*real(Nu);

    % impose BC and reshape into matrix
    W(:,:,i+1) = padarray(reshape(V0_vec + 1,M-2,M-2),[1 1],1);

end
```

The Lanczos ETDRK4 Algorithm

```
%
% Author: Jeffrey Allen
%
% solves the 2D Allen Cahn equation via ETDRK4 time stepping and
% second order centered differences space discretization
%
% u_t = epsilon*(u_xx + u_yy) + F(u) on [-1,1]x[-1,1],
%
% where F(u) = u-u^3, and Dirichlet BC's with u=1
%
% ETDRK4 coefficients are computed using the Lanczos approximation
%
% Script requires the functions: laplacian(), expml(), lanczos2(),
% and padm()

% poles, weights for r=s=2
c1 = -3.0 + 1.732050807568877293 52*1i;
w1 = -6.0 - 10.392304845413263761 1*1i;
w11 = -0.5 - 1.443375672974064411 27*1i;
w21 = -1.154700538379251529 01*1i;
w31 = 0.5 + 0.288675134594812882 25*1i;
tdc1 = -6.0 + 3.464101615137754587 0548*1i;
tdw1 = -12.0 - 20.784609690826527522 32935*1i;
omega1 = -3.464101615137754587 05*1i;

% parameter
epsilon = 0.01;

% number of grid points per dimension
M = 50;

% 1D meshes
x = linspace(-1,1,M)';
y = x;

% 2D mesh
[X,Y] = meshgrid(x,y);

% spatial step
dx = 2/(M-1);

% initial data
U = 1 - 2*exp(-10*((X+.35).^2 + (Y+.35).^2)) - ...
```

```
        2*exp(-18*((X-.40).^2 + (Y-.40).^2)) - ...
        2*exp(-15*((X-.25).^2 + (Y+.25).^2));

% constructs discretized 2D Laplacian matrix
B = {'DD','DD'};
[~,~,A] = laplacian([M-2 M-2],B);
A = (epsilon/(dx^2))*A;
I = speye(size(A));

% time step
k = .01;

% maximum iterations
Tmax = 5; Nmax = round(Tmax/k);

% homogenize
V = U - 1;

% (M-2)^2 unknowns
V0 = V(2:M-1,2:M-1);

% solution matrix
W = zeros(M,M,Nmax);

% impose IC and BC
W(:,:,1) = padarray(V0 + 1,[1 1],1);

% column ordering
V0_vec = V0(:);

% krylov subspace dimension
m = 25;

% LHS matrix for (2,2)-Pade
M1 = k*A - c1*I;

% Cholesky decomp
L = chol(A,'lower');

% time stepping
for i = 1:Nmax

    Fv = (V0_vec+1) - (V0_vec+1).^3;
    z = expml(A,A*V0_vec - Fv,m,-k/2);
    z = z + Fv;
```

```
    y = L\z;
    a = L'\y;


    Fa = (a+1) - (a+1).^3;
    z = expml(A,A*V0_vec - Fa,m,-k/2);
    z = z + Fa;
    y = L\z;
    b = L'\y;

    Fb = (b+1) - (b+1).^3;
    z = expml(A,A*a - 2*Fb + Fv,m,-k/2);
    z = z + 2*Fb - Fv;
    y = L\z;
    c = L'\y;

    Fc = (c+1) - (c+1).^3;
    Nu = M1\(w1*V0_vec + k*w11*Fv + k*w21*(Fa + Fb) + k*w31*Fc);
    V0_vec = V0_vec + 2*real(Nu);

    % impose BC and reshape into matrix
    W(:,:,i+1) = padarray(reshape(V0_vec + 1,M-2,M-2),[1 1],1);

end
```