The University of Nottingham

School of Mathematical Sciences

Division of Applied Mathematics

Theoretical Mechanics

Numerical Methods for Stiff Systems

A thesis submitted to the University of Nottingham for the Degree of Doctor of Philosophy

by

Hala Ashi

Supervisors

Dr. Paul Matthews

Dr. Linda Cummings

June 2008

Abstract

Some real-world applications involve situations where different physical phenomena acting on very different time scales occur simultaneously. The partial differential equations (PDEs) governing such situations are categorized as "stiff" PDEs. Stiffness is a challenging property of differential equations (DEs) that prevents conventional explicit numerical integrators from handling a problem efficiently. For such cases, stability (rather than accuracy) requirements dictate the choice of time step size to be very small. Considerable effort in coping with stiffness has gone into developing time-discretization methods to overcome many of the constraints of the conventional methods. Recently, there has been a renewed interest in exponential integrators that have emerged as a viable alternative for dealing effectively with stiffness of DEs.

Our attention has been focused on the explicit **Exponential Time Differencing** (ETD) integrators that are designed to solve stiff semi-linear problems. Semi-linear PDEs can be split into a linear part, which contains the stiffest part of the dynamics of the problem, and a nonlinear part, which varies more slowly than the linear part. The ETD methods solve the linear part exactly, and then explicitly approximate the remaining part by polynomial approximations.

The first aspect of this project involves an analytical examination of the methods' stability properties in order to present the advantage of these methods in overcoming the stability constraints. Furthermore, we discuss the numerical difficulties in approximating the ETD coefficients, which are functions of the linear term of the PDE. We address ourselves to describing various algorithms for approximating the coefficients, analyze their performance and their computational cost, and weigh their advantages for an efficient implementation of the ETD methods.

The second aspect is to perform a variety of numerical experiments to evaluate the usefulness of the ETD methods, compared to other competing stiff integrators, for integrating real application problems. The problems considered include the **Kuramoto-Sivashinsky** equation, the nonlinear **Schrödinger** equation and the nonlinear **Thin Film** equation, all in one space dimension. The main properties tested are accuracy, start-up overhead cost and overall computation cost, since these parameters play key roles in the overall efficiency of the methods.

To the ever living memory of my father and my father-in-law

"May ALLAH bestow His mercy upon them and award them paradise"

Acknowledgments

First and foremost, I would like to thank ALLAH for giving me the strength and making the completion of this thesis possible.

Early in this project, when I took the preliminary steps towards my thesis, I did not foresee how many individuals would contribute to its completion whom I owe my thanks and appreciation to. I am particularly grateful to my supervisors, Dr. Paul Matthews and Dr. Linda Cummings, for their unlimited support and guidance, thoughtful suggestions, objective comments, practical advice, and insightful direction. Without their knowledge and expertise this thesis would never have been completed.

I would like to express my sincere appreciation to Helen Cunliffe, the Research Secretary, for her continuous support and assistance.

My gratitude also goes to my beloved husband, Mamdouh Tayeb, whose boundless love and firm belief in me have been a source of inspiration for me during my lengthy project.

I am extremely motivated to express my greatest appreciation to my mother and mother-in-law for their continuous support and prayers for me.

Finally, special mention must be made of my children Taher, Mohammed, and Tala, I love you all.

Contents

Chapter 1

Introduction

1.1 Introduction

The numerical solution of ordinary differential equations (ODEs) is an old topic. Various techniques have been devised over the years to solve such equations, and astonishingly, the old well-established methods such as the Runge-Kutta methods [78] are still the foundation for the most effective and widely-used codes. Nevertheless, there are several kinds of problems which classical methods do not handle very effectively, problems that are said to be "**stiff**".

The earliest detection of stiffness in differential equations in the digital computer era, by the two chemists, **Curtiss** and **Hirschfelder** (1952) [20], was apparently far in advance of its time. They named the phenomenon and spotted the nature of stiffness (stability requirement dictates the choice of the step size to be very small). To resolve the problem they recommended possible methods such as the **Backward Differentiation Formula** [78] for numerical integration. In 1963, **Dahlquist** [21] defined the problem and demonstrated the difficulties that standard differential equation solvers have with stiff differential equations.

At about this time several authors participated in independent research for handling and evading the problems posed by stiff differential equations. For example, **Gear** [31] in 1968, became one of the most important names in this field. More articles on integrating stiff ODEs are listed in [44, 68].

Considerable efforts have gone into developing numerical integration for stiff problems [72], and hence, the problem of stiffness was brought to the attention of the mathematical and computer science community, see [33] for further details on the topic of stiffness, and [30] for a comprehensive review of this phenomena.

Stiff differential equations are categorized as those whose solutions (or different components of a single solution) evolve on very different time scales occurring simultaneously, i.e. the rates of change of the various components of the solutions differ markedly. Consider, for example, if one component of the solution has a term of the form e^{-ct}, where c is a large positive constant. This component, which is called the transient solution, decays to zero much more rapidly, as t increases, than other slower components of the solutions. Alternatively, consider a case where a component of the solution oscillates rapidly on a time scale much shorter than that associated with the other solution components.

For a numerical method which makes use of derivative values, the fast component continues to influence the solution, and as a consequence, the selection of the step size in the numerical solution is problematic. This is because the required step size is governed not only by the behavior of the solution as a whole, but also by that of the rapidly varying transient which does not persist in the solution that we are monitoring.

In reality, the numerical values occurring in nature are frequently such as to cause stiffness. Therefore, a realistic representation of a natural system using a differential equation is likely to encounter this phenomenon. An example is the field of chemical kinetics [20]. Here ordinary differential equations describe reactions of various chemical species to form other species. The stiffness in such systems is a consequence of the fact that different reactions take place on vastly different time scales.

Another important class of stiff ODEs originates frequently from application of the general approach *'the method of lines'* [84] to stiff time-dependent PDEs. In this method we first discretize the spatial derivatives of a PDE with a spatial derivative approximation method, which results in a stiff coupled system of ordinary differential equations (ODEs) in time only. Then, we apply any well established numerical method to achieve an accurate approximate solution to the problem. Two broadly applicable techniques include **Finite Difference Formulas** [58, 83], which are local methods and **Spectral Methods** [25, 83, 84], which are global methods, see §2 for further details.

The idea of using spectral representations for numerical solutions of ODEs goes back at least to **Lanczos** [51] in 1938. Spectral methods are a class of techniques used in applied mathematics and scientific computing to numerically solve certain partial differential equations, often involving the use of the **Fast Fourier Transform (FFT)** algorithm of **Cooley** and **Tukey** [18]. A short historical summary of the FFT can be found in [12], while a comprehensive survey and its mathematical applications can be found in **Henrici**'s article [34]. Spectral methods have been widely used for spatial discretization in the context of solving time-dependent PDEs since the early 1970's, see for example the article published in 1972 by **Kreiss** and **Oliger** [48]. The books by **Trefethen** [84] and **Fornberg** [25] are intended for researchers interested in exploring this field of study.

If the differentiation matrix applied to discretize the spatial derivatives has eigenvalues with very diverse values, i.e. the ratio of largest to smallest (in magnitude) eigenvalues is very large, or if a PDE has spatial derivatives of higher than second order, then the problem is more likely to be stiff. The degree of stiffness depends on the grid spacing of the spatial discretization. As we decrease the grid spacing, i.e. increase the number of points with which we are discretizing the operator, the eigenvalues vary greatly in magnitude.

Given that stiffness has extensive practical applications and arises in many physical situations, the demand for special techniques that permit the use of a step size governed only by the rate of change of the overall solution is very great.

Despite the fact that numerical integrations of stiff systems with constant coefficients have been considered in detail, a stiff differential equation does not lend itself readily to numerical solution by classical methods [33]. In principle, the stability region of the integration method must include the eigenvalues of the discrete linear operator of a stiff PDE in order to be stable. Linear explicit schemes have a penalty of requiring an extremely small step size in order to be stable, causing unacceptable increase in the number of integration steps and in the integration times and even an excessive error accumulation. On the other hand, implicit schemes have an advantage of the freedom of choice of the time step and nice stability properties. However, discretization of a nonlinear PDE leads to a large nonlinear system of equations that has to be solved at each time step. This renders implicit schemes costly to implement. Thus, the goal of developing more efficient time integrators for stiff systems is to provide alternative choices of more sophisticated schemes that have better stability properties and require fewer arithmetic operations per time step than standard explicit and implicit schemes respectively.

Various methods have been proposed to avoid the difficulties that appear when trying to solve nonlinear equations with an implicit method. A popular strategy is to combine pairs of an explicit multi-step formula to advance the nonlinear part of the problem and an implicit method to advance the linear part. This strategy forms the basis of the so-called **Implicit-Explicit (IMEX)** schemes. These schemes were proposed to solve stiff PDEs as far back as the late 1970's [87]. The direct derivation of the linear-multi-step IMEX schemes and their stability properties is fully documented in a paper by **Ascher** [4], and further stability analysis can be found in [27]. Other more complicated forms of the IMEX schemes such as the **Runge-Kutta** IMEX schemes are reported in [3, 16].

The most popular second-order linear-multi-step IMEX scheme (**AB**2**AM**2) [4] utilizes the second-order **Adams-Moulton** [14] and **Adams-Bashforth** [14] schemes to advance the linear and the nonlinear parts of the problem in time respectively. Unfortunately, it is not always so easy to construct an IMEX scheme by coupling two multi-step methods. From an accuracy point of view, the combination of the order of accuracy of each coupled implicit and explicit methods must give the correct order of accuracy of the overall method.

IMEX schemes can be useful methods, especially when used in conjunction with spectral methods, for approximating spatially discretized reaction-diffusion problems [66] arising in chemistry and mathematical biology. For these problems the nonlinear reaction term can be treated explicitly while the diffusion term is treated implicitly. Examples for reaction-diffusion systems from a biological standpoint can be found in [62].

IMEX methods are restricted from having an order higher than two if A-stability[1] is required (this is the second Dahlquist stability barrier [78]). Therefore, despite their simplicity and frequent usage, they are not extendable to higher order. A subset of these schemes are the backwards differencing schemes. These schemes are frequently used in stiff problems because, although they may not be A-stable for order greater than two, they do correctly damp non-oscillatory decaying perturbations (but not those which are oscillatory in general) [43].

Nonlinear methods or methods with non-constant coefficients are not restricted by the Dahlquist Barrier and may be generalized to arbitrary order. Such schemes have been explored by several authors [37, 82] to solve stiff DEs. In 1960, **Certaine** [17] observed that the negligible rapidly varying transient solution, in a system of first-order coupled differential equations, is a hindrance to any conventional scheme to give an accurate solution to the system (an example is the **Trapezium Rule** [14]). He resolved this by coming up with a new class of nonlinear schemes based on the **Adams-Moulton** methods of second and third order. A distinctive feature of these schemes is the exact evaluation of the linear part of the differential equation (and so the schemes are automatically A-stable). That is, if the nonlinear part is zero, then the scheme reduces to the evaluation of the exponential function of the operator (or matrix) that represents the linear part.

Historically, various constructions of **Certaine** schemes with various names have been derived since the 1960's. In 1969, **Nørsett** [63] modified the **Adams-**

[1] *A-stability* is the property that physically decaying solutions are numerically damped for any choice of time step. This feature is highly desirable for stiff problems, as fast decaying perturbations would be damped even with time steps much longer than their life time.

Bashforth formulas to be A-stable methods of arbitrary order, suitable for the numerical integration of a stiff system of ODEs. In 1998, **Beylkin** *et al.* [9] constructed implicit and explicit schemes of arbitrary order, which they called **Exact Linear Part (ELP)** method. They analyzed, in detail, the stability of the methods when applied to solve nonlinear PDEs. However, the formulas of the methods' coefficients were not given explicitly. Later in 2002, a clear derivation of the explicit ELP schemes of arbitrary order was given by **Cox** and **Matthews** [19], where they referred to these methods as the '**Exponential Time Differencing (ETD)**' methods (the term used arose originally in the field of computational electrodynamics [40, 65, 71]). The authors also broadened these schemes to more accurate ETD '**Runge-Kutta**' (ETD-RK) methods, and illustrated the superior performance of the ETD methods when they were applied to both *dissipative* and *dispersive* PDEs. Furthermore, a class of exponential propagation techniques known as **Exponential Propagation Iterative (EPI)** schemes were introduced by **Tokman** in [81]. These schemes were constructed by reformulating the integral form of a solution to a nonlinear autonomous system of ODEs as an expansion in terms of products of matrix and vector functions. To trace the history of discovering and developing the ETD schemes see [89].

The formula of the ETD schemes is based on integrating the linear parts of the differential equation exactly, and approximating the nonlinear terms by a polynomial, which is then integrated exactly. A similar approach is used in the **Integrating Factor (IF)** schemes, which were introduced first in 1967 by **Lawson** [52]. In the approach of the IF schemes [7, 11, 19, 44, 45, 49, 84], we multiply both sides of an ODE by an appropriate integrating factor and obtain a differential equation in which we change variables so that the linear part can be solved exactly. Afterwards, we apply any numerical scheme (multi-step or Runge-Kutta methods) to integrate the transformed nonlinear equation. Then we transform back the approximated solution which becomes an approximate solution for the original variable, see [8, 57] for a comprehensive review.

Methods like ETD and IF, based on the exact treatment of the linear terms, require the computation of matrix exponentials (or matrix functions) for the linear operators. However, as pointed out by **Cox** and **Matthews** [19] in their implementation of the ETD methods, a computational problem arises when evaluating the methods' coefficients. When we discretize a stiff semi-linear PDE in space, the linear operator of the resulting system of coupled ODEs, which is represented by a diagonal (in case of discretizing with Fourier spectral methods) or a non-diagonal matrix (in case of discretizing with finite difference formulas or Chebyshev polynomials [11, 25, 83, 84]), might have zero, large and small (in magnitude) eigenvalues. For eigenvalues approaching zero, the ETD coefficients give imprecise results because of the large amount of cancellation in their formulas. This problem gets worse with the higher order ETD methods. For eigenvalues equal to zero, we are actually dividing the numerator by zero denominator in the explicit formulas for the coefficients, which renders them useless in this case. To deal with the problem in the case where the linear operator is represented by a diagonal matrix, the authors of [19] used the Taylor series to evaluate such coefficients for small eigenvalues and used the explicit formulas of the ETD coefficients for large eigenvalues. However, this process cannot be applied in case of the linear operator being a non-diagonal matrix because the eigenvalues of small and large magnitude are indistinguishable. It is therefore important to have an accurate numerical algorithm for evaluating the ETD coefficients.

We note that the problem of computing the exponential of large matrices has been of interest in numerical analysis for a long time [59]. Recently, **Moler** and **Van Loan** [60] updated the publication of "Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later", in which they analyzed the efficiency of various algorithms and gave further developments in computing a matrix exponential. One example is the algorithm that is based on scaling and squaring which proved its efficiency in approximating a matrix exponential accurately [9]. Additionally, various algorithms have been devoted to compute non-diagonal matrix functions efficiently by many authors [2, 8, 35, 37, 47, 53, 54, 56, 57, 67, 70, 76, 80, 81]. Recently, **Kassam** and **Trefethen** [44, 45] proposed a modification of the ETD **Runge-Kutta** schemes of **Cox** and **Matthews** [19], to ameliorate the numerical difficulties associated with these schemes. The key idea is to evaluate the ETD coefficients by means of contour integrals in the complex plane using the well-known **Cauchy Integral Formula** [55]. Further discussion of this issue is detailed in §4.

Exponential Time Differencing methods have extensive applications in solving stiff systems [15, 39]. For example, in the field of chemical kinetics problems, the author of [81] conducted some numerical comparisons in which he deduced that explicit exponential integrators are highly competitive relative to those standard integrators. The authors of [22, 23] indicated that the higher order ETD based schemes can be several orders of magnitude faster than low-order semi-implicit methods in some simulations of micro-structure evolution (a core component of phase field modeling) in two and three dimensions. Moreover, **Kassam** and **Trefethen** [44, 45] compared various fourth-order methods, including the ETD methods of [19], and their results revealed that the best is the ETD4RK method of [19] for solving various one-dimensional diffusion-type problems. However, more recently **Krogstad** [49] presented a fourth-order ETDRK4-B method with better accuracy than the ETD4RK method of [19] and illustrated its efficiency in solving several semi-discretized PDEs, such as the Kuramoto-Sivashinsky (K-S) equation [41]. A recent report [57] also showed that the ETD type of exponential integrators surpass integrators of Lawson type [52] in solving parabolic semi-linear problems, such as the K-S and the nonlinear Schrödinger (NLS) equations [77]. Again and under certain circumstances, **Berland** and **Skaflestad** [7] solved the NLS equation and found that the performance of a fourth-order Lawson integrating factor method was demonstrably poorer than the fourth-order ETD4RK method of [19].

Related work in numerical simulations of stiff problems has made extensive use of the ETD methods [8, 37, 46]. The explicit ETD methods have proved their efficiency in numerous applications, for example, in computational electrodynamics [71], in reaction kinetics [61] and in solving incompressible magnetohydrodynamics equations [53]. This further motivates our theoretical and numerical investigation on the various properties of the ETD schemes while carrying out more computational studies for real application problems.

1.2 Layout Of Thesis

The main objective of this thesis is to present the **Exponential Time Differencing** (ETD) schemes as a viable alternative to classical integrator methods for solving stiff semi-linear PDEs. In semi-linear PDES, the linear part contains higher order spatial derivatives than those in the nonlinear part. We place emphasis on the stability, accuracy, efficiency and reliability of these numerical integrators.

The purpose of chapter 2 is to present the "**Method of Lines**" procedure for solving initial boundary value problems. This procedure starts with discretizing the spatial derivatives in the PDE with algebraic approximations. We include two spatial derivative approximation techniques: **Finite Difference Formulas** and **Spectral Methods**. We show through examples how to formulate the resulting semi-discrete problem, which is a stiff system of coupled ODEs with time as the only independent variable.

The next step in the procedure is to search for an accurate and fast numerical method and apply it to these initial value ODEs to compute an approximate numerical solution to the PDE. Hence, we consider in chapter 3, the ETD schemes of arbitrary order as time-discretization methods. We give in detail the derivation of these methods following the approach in [9, 17, 19, 63], and present the ETD-RK type constructed in [19]. In addition, we examine analytically the methods' stability properties to present the advantage of these methods in overcoming the stability constraints that are imposed on any conventional explicit method utilized to solve a stiff system of ODEs. The approach is to compute the boundaries of the stability regions in two dimensions for a general test problem, where the stiffness parameter is negative and purely real.

In chapter 4, we go through the difficulties occurring in the computation of the ETD coefficients (as mentioned previously in the introduction, the evaluation of coefficients for eigenvalues approaching zero suffers from numerical rounding errors due to the large amount of cancellation in the explicit formulas). We conduct comparison experiments on various algorithms and analyze their performance and their computational cost for an accurate evaluation of the coefficients and an efficient implementation of the ETD methods. The algorithms included are the Taylor series, the Cauchy integral formula, the Scaling and Squaring algorithm, the Composite Matrix algorithm and the Matrix Decomposition algorithm for non-diagonal matrix cases. The matrices considered are the second-order centered difference differentiation matrix for the first and second derivatives and the Chebyshev differentiation matrix for the second derivative to show that the algorithms' efficiency is by no means restricted to any special structure of certain matrices. We have published in the article [5] (in press) some of the results developed in this chapter.

In chapter 5, we demonstrate the effectiveness of the ETD methods for integrating real application problems. For the simulation tests, we consider three one space-dimensional problems with periodic boundary conditions. We apply Fourier spectral approximation for the spatial discretization, and employ first, second and fourth-order ETD methods, first-order Implicit-Explicit (IMEX) method and first, second and fourth-order Integrating Factor (IF) methods for time discretization. The first two problems considered are the time-dependent scalar **Kuramoto-Sivashinsky (K-S)** equation and the nonlinear **Schrödinger (NLS)** equation. In these two equations, the linear terms are primarily responsible for stiffness. The stiffness in the K-S equation is due to the strong dissipation of high wave number modes on a time scale much shorter than that typical of the nonlinear term, whereas, the stiffness in the NLS equation is due to the rapid oscillations of high wave number modes. The third model considered is the nonlinear **Thin Film** equation. Solving this equation is a more challenging task since the nonlinear terms are the ones responsible for stiffness. To facilitate numerical studies of the thin film equation, we consider a perturbation to the uniform solution of the equation and obtain after a few algebraic manipulations two split parts of the linear and nonlinear terms. The stiffness in the problem is again due to the strong dissipation.

The main testing factors in differentiating between the methods are the stability, the accuracy, the start-up overhead cost and the CPU time consumed by the methods. To address the question of stability and accuracy of the methods we perform a series of runs with different choices of final times which are computed, for all methods, with various time step sizes. The time step values are selected to ensure that all methods achieve stable accurate results. We measure the accuracy in terms of the relative error evaluated in the integrated error norm. Then, we turn our attention to the accuracy of the methods as a function of CPU time. All the calculations presented in this chapter are performed using Matlab codes. We evaluate the coefficients of the ETD methods, once at the beginning of the integration for each value of the time step sizes, using the 'Cauchy integral' approach proposed by **Kassam** and **Trefethen** [44, 45].

Finally, in chapter 6, we conclude with a brief discussion of the work carried out and the main results drawn from this research and reiterate the main conclusions; additionally, we outline a number of possible extensions to this work in further future studies.

Chapter 2

Spatial Discretization Methods

Outline of Chapter

Our physical world is most generally described in scientific terms with respect to three space dimensions and time. Time-dependent partial differential equations (PDEs) provide a mathematical description for a large range of physical space-time problems, and are very widely used in applied mathematics.

A general numerical procedure for solving initial boundary value problems is the "**Method of Lines**". This procedure starts with discretizing the spatial derivatives in the PDE with algebraic approximations. The resulting semi-discrete problem, which is a system of coupled ordinary differential equations (ODEs) with time as the only independent variable, must then be integrated. The method of lines is an efficient tool that allows standard (accurate) general methods that have been developed for the numerical integration of ODEs to be used.

The purpose of this chapter is to present two spatial derivative approximation techniques: **Finite Difference Formulas** [58, 83] and **Spectral Methods** [25, 83, 84], and show, through examples, how to formulate the system of ODEs that approximates the original PDE.

2.1 Introduction

The field of partial differential equations (PDEs) is broad and varied, as is inevitable because of the great diversity of physical phenomena that these equations model. Much of the variety is introduced by the fact that practical problems involve different geometric classifications (hyperbolic, elliptic, parabolic), multiple space dimensions, systems of PDEs, different types of boundary conditions, varying smoothness of the initial conditions, variable coefficients and frequently, nonlinearity.

A well-known numerical approach to solve a time-dependent PDE, whose solutions vary both in time and in space, is the **Method of Lines** [84]. In the approach of this method, firstly, we construct a semi-discrete approximation to the problem by setting up a regular grid in space[1], i.e. the spatial independent variables that have boundary constraints are discretized. Thereby, we generate a coupled system of ordinary differential equations (ODEs) in the time independent variable t, which is associated with the initial value. Secondly, we numerically approximate solutions to the original PDE by marching forward in time on this grid. Now we can apply any existing, and generally well established, numerical methods (such as the Exponential Time Differencing methods, see §3 for more details) for these initial value ODEs to compute an approximate numerical solution to the PDE.

The idea of semi-discretization focuses attention on spatial difference operators as approximations of spatial differential operators. Two broadly applicable spatial derivative approximation techniques are **Finite Difference Formulas** [58, 83] and **Spectral Methods** [25, 83, 84]. The key factors in selecting among these techniques are the nature of the grid, the complexity of the domain and the required levels of accuracy of differentiation. These techniques are discussed in §2.2 and §2.3 respectively.

2.2 Finite Difference Formulas

The purpose of discretizing time-dependent PDEs is to obtain a problem that can be solved by a finite procedure. The simplest such kind of finite procedure is the Finite Difference Formula (FDF). A FDF is a fixed formula that approximates a

[1]When solving a one-dimensional time-dependent PDE, we assume (throughout the thesis) a fixed space step $h > 0$ and a fixed time step $\Delta t > 0$ for discretizing the spatial part x and temporal part t respectively. Thereby, we are defining the points (x_n, t_j), for any integers n and j, in a two dimensional regular grid, formally, the subset $h\mathbb{Z} \times \Delta t\mathbb{Z}$ of $\mathbb{R}^2$.

continuous function by a function of a finite number of grid values; thereby, we obtain a finite system of equations to be solved.

In this section we describe FDFs as discrete approximations to the spatial derivatives of a PDE. Given a function on a set of uniform grid points, Finite Difference Approximations (FDAs) approximate the derivative of the function by the derivative of the local interpolators on the grid [84].

2.2.1 Finite Difference Approximation

Suppose that a function $f(x)$, defined on an interval $0 < x < L$ that is divided into q subintervals, has some known values at a finite number of points on a uniform mesh of size $h = L/q$. FDAs approximate the first and second numerical derivatives $df(x)/dx$ and $d^2f(x)/dx^2$ of $f(x)$ respectively for example, as follows,

$$\left.\frac{df(x)}{dx}\right|_{x=x_n} = \frac{f(x_{n+1}) - f(x_n)}{h} + O(h),$$

and

$$\left.\frac{d^2f(x)}{dx^2}\right|_{x=x_n} = \frac{f(x_n) - 2f(x_{n+1}) + f(x_{n+2})}{h^2} + O(h).$$

The two approximations above are derived using the Taylor series and are called **Forward Differentiation**. Note that the first derivative $df(x)/dx$ is obtained using the values of $f(x)$ at the points x_n and x_{n+1}, and the second derivative uses the points x_n, x_{n+1} and x_{n+2}. These approximations have a truncation error (obtained from truncating the Taylor series) of $O(h)$, that is local to the interval enclosing the sampling points. For sufficiently small h, the errors are then proportional to h, and the approximations are **first-order** in h.

When using finite differences, it is important to keep in mind that there are several sources of errors: the truncation error (which is introduced by truncating the Taylor series approximation), roundoff error and condition error. Roundoff error comes from roundoff in the arithmetic computations required. Condition error comes from magnification of any errors in the function values. It arises typically from the division by a power of the step size, and so grows with decreasing step size. This means that in practice, even though the truncation error approaches zero as h does, the actual error starts growing beyond some point.

To obtain higher-order approximations to the derivative, it is easy to invoke many function values further away from the point of interest. Thus, the **second-**

order forward finite difference approximation for $df(x)/dx$ is

$$\left.\frac{df(x)}{dx}\right|_{x=x_n} = \frac{-3f(x_n) + 4f(x_{n+1}) - f(x_{n+2})}{2h} + O(h^2),$$

and for $d^2f(x)/dx^2$ is

$$\left.\frac{d^2f(x)}{dx^2}\right|_{x=x_n} = \frac{2f(x_n) - 5f(x_{n+1}) + 4f(x_{n+2}) - f(x_{n+3})}{h^2} + O(h^2).$$

The simplest finite difference approximations are centered and symmetrical, i.e. they use values of the function at points either side equally and always have even order of accuracy. Using the Taylor expansions for $f(x_{n+1})$ and $f(x_{n-1})$, the **second-order centered finite difference approximation** for $df(x)/dx$ is

$$\left.\frac{df(x)}{dx}\right|_{x=x_n} = \frac{f(x_{n+1}) - f(x_{n-1})}{2h} + O(h^2),$$

and for $d^2f(x)/dx^2$ is

$$\left.\frac{d^2f(x)}{dx^2}\right|_{x=x_n} = \frac{f(x_{n+1}) - 2f(x_n) + f(x_{n-1})}{h^2} + O(h^2).$$

Note that it is not possible to use centered finite difference approximations in some cases as follows:

- If we are near the boundary, required values of $f(x)$ may not be available.
- For certain problems, the stability is improved with one-sided forward finite differences,

and hence one-sided forward finite difference approximations are sometimes used instead.

In general, formulas for any given derivatives of any chosen order can be derived from Taylor expansions as long as a sufficient number of sample points are used. However, these approximations become cumbersome beyond the simple examples shown above. We refer the reader to the book by **Fornberg** [25], where the centered and one-sided Finite Differences formulas for approximating derivatives up to fourth-order, for equi-spaced grids, with order of accuracies up to the eighth are readily available in tables.

2.2.2 An Example

As a basic illustrative example of a PDE, we consider the linear heat (diffusion) equation

$$\frac{\partial u(x,t)}{\partial t} = \nu \frac{\partial^2 u(x,t)}{\partial x^2}, \qquad t_0 \le t \le T,\ x_0 \le x \le x_q, \tag{2.1}$$

where

- $u(x,t)$ is the dependent variable,
- t and x are the independent variables representing time and one-dimensional space respectively,
- ν is a real positive constant,

subject to an initial condition at $t_0 = 0$

$$u(x, t = 0) = u^0(x),$$

and two boundary conditions, corresponding to boundaries of a physical system (there are other possibilities)

$$u(x = x_0, t) = f(t), \; u(x = x_q, t) = g(t),$$

where $f(t)$ and $g(t)$ are given boundary values of u for all t.

To illustrate the method of lines procedure to solve the heat equation (2.1), suppose that $u(x,t)$ is discretized in space with $q+1$ points, of which $q-1$ are interior points, on a uniform grid with step size h as follows

$$u(x_n, t) \approx u_n(t), \qquad 0 \le n \le q,$$

where the index n designates a position along the grid in x. To approximate the spatial derivative $\partial^2 u/\partial x^2$ in equation (2.1), we use for example, the second-order centered finite difference approximation

$$\left.\frac{\partial^2 u(x,t)}{\partial x^2}\right|_{x=x_n} \approx \frac{u_{n+1}(t) - 2u_n(t) + u_{n-1}(t)}{h^2} + O(h^2). \tag{2.2}$$

Substituting equation (2.2) into (2.1), gives a system of $q-1$ approximating ODEs

$$\begin{aligned} u_0(t) &= f(t), \\ du_1(t)/dt &= \nu(u_2(t) - 2u_1(t) + u_0(t))/h^2, \\ du_2(t)/dt &= \nu(u_3(t) - 2u_2(t) + u_1(t))/h^2, \\ &\vdots \\ du_{q-1}(t)/dt &= \nu(u_q(t) - 2u_{q-1}(t) + u_{q-2}(t))/h^2, \\ u_q(t) &= g(t), \end{aligned} \tag{2.3}$$

subject to the initial conditions

$$u_n(t = 0) = u^0(x_n), \qquad 0 \le n \le q. \tag{2.4}$$

To complete the solution of the original PDE (2.1), we compute a solution to the approximation system of ODEs (2.3). The system (2.3) and the initial conditions (2.4) now constitute the complete method of lines approximation of equation (2.1).

2.2.3 Matrix Form

Since differentiation and finite difference approximation are linear operations, an alternative way of representing an approximation to the differential operator is with a matrix. This matrix is referred to as a differentiation matrix.

Using second-order centered FDAs, for example, to approximate the spatial derivatives on a uniform grid of $q+1$ points, reduces the problem to $q-1$ coupled ODEs. Hence, for any given non-periodic boundary conditions, the differentiation matrix representing the second derivative, for example, is a $(q-1)\times(q-1)$ **tridiagonal** matrix of the form

$$M_2 = \frac{1}{h^2}\begin{pmatrix} -2 & 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & -2 \end{pmatrix}.$$

For periodic boundary conditions, M_2 is of the same form but has a 1 in the top right and bottom left corners. Similarly, for the fourth-order centered approximation, the resulting differentiation matrix, representing the second derivative is **pentadiagonal**. As a result, the order of the approximation determines the sparsity of the matrix.

2.3 Spectral Methods

In spectral methods, instead of representing a function by its values at grid points, the function is written as an expansion in terms of smooth basis functions $\phi_k(x)$ as

$$f(x) \approx \sum_{k=1}^{N} a_k \phi_k(x), \tag{2.5}$$

for some integer N.

Spectral methods are global approximations since the values of the spectral coefficients a_k influence the function and its derivatives for all x, whereas finite difference methods are local approximations since the value $f(x_n)$ of $f(x)$ at a grid point x_n only has influence near that point.

The definitive advantage of spectral methods lies in their remarkable accuracy properties. For any given analytic function, spectral approximations approximate the function and its derivatives within an exponential accuracy. It was shown [79] that when differencing analytic functions on regular grids using spectral methods, the errors decay to zero at an exponential rate as the grid is refined, rather than at (much slower) polynomial rates obtained by finite difference formulas. This behavior is essentially due to the corresponding exponential decay of the spectral coefficients as the number of grid points is increased.

The basis functions $\phi_k(x),\ k = 1, \ldots, N$ used in the expansion (2.5) must satisfy three criteria [25]:

(i) The coefficients a_k must decrease rapidly with k, which ensures the rapid convergence of the approximation (2.5) of $f(x)$.

(ii) It should be easy to write the first derivative $df(x)/dx$ as an expansion in the same basis functions with the coefficients b_k such that

$$\frac{d}{dx}\left(\sum_{k=1}^{N} a_k\phi_k(x)\right) = \sum_{k=1}^{N} b_k\phi_k(x),$$

for given coefficients a_k.

(iii) One must be able to convert from the coefficients a_k in the 'spectral' space and the function $f(x)$ in 'physical' space.

Usually, each spectral method is named after choosing a function class for the basis functions. For non-periodic problems, the preferred choice is the orthogonal[2] **Chebyshev polynomials** and the method is referred to as the **Chebyshev Spectral** method. **Boyd** [11] and **Trefethen** [83] are valuable references for this case.

[2]The set of functions $\{\phi_0, \phi_1, \ldots \phi_n\}$, defined on an interval $[a, b]$, is said to be an **orthogonal set**, with respect to the weight function w, if

$$\int_a^b w(x)\phi_j(x)\phi_k(x)\ dx = \begin{cases} 0 & \text{for } j \neq k, \\ \alpha_k > 0 & \text{for } j = k. \end{cases}$$

for some constant α_k. In addition, if $\alpha_k = 1$ for each $k = 0, 1, \ldots, n$, the set is said to be **orthonormal**.

For periodic problems, the natural choice of basis functions is the *trigonometric functions*, and the function is ideally represented by its **Fourier series**. Here the method is referred to as the **Fourier Spectral** method [83, 84].

2.3.1 Fourier Spectral Methods

Fourier analysis occurs in the modeling of time-dependent phenomena that are exactly or approximately periodic. Examples of this include the digital processing of information such as speech; the analysis of natural phenomena such as earthquakes; in the study of vibrations of spherical, circular or rectangular structures; and in the processing of pictures. In a typical case, Fourier spectral methods write the solution to the PDE as its Fourier series. Fourier series decomposes a periodic real-valued function of real argument into a sum of simple oscillating trigonometric functions (*sines* and *cosines*) that can be recombined to obtain the original function. Substituting this series into the PDE gives a system of ODEs for the time-dependent coefficients of the trigonometric terms in the series (this series is usually written in complex exponential form); then we choose a time-stepping method to solve those ODEs.

1. **Fourier Series**.

   The Fourier series of a smooth and periodic real-valued function $f(x) \in C[0, 2L]$ with period $2L$ is

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(n\pi x/L) + b_n \sin(n\pi x/L)).$$

   Since the basis functions $\cos(n\pi x/L)$ and $\sin(n\pi x/L)$ are orthogonal, i.e.

$$\begin{aligned} \int_0^{2L} \cos(n\pi x/L) \sin(m\pi x/L)\ dx &= 0, \\ \int_0^{2L} \cos(n\pi x/L) \cos(m\pi x/L)\ dx &= L\delta_{mn}, \\ \int_0^{2L} \sin(n\pi x/L) \sin(m\pi x/L)\ dx &= L\delta_{mn}, \end{aligned}$$

   where

$$\delta_{mn} = \begin{cases} 0 & \text{for } m \neq n, \\ 1 & \text{for } m = n, \end{cases}$$

   the coefficients are given by

$$a_n = \frac{1}{L} \int_0^{2L} f(x) \cos(n\pi x/L)\ dx, \qquad \text{for each } n = 0, 1, \ldots,$$

and

$$b_n = \frac{1}{L}\int_0^{2L} f(x)\sin(n\pi x/L)\,dx, \qquad \text{for each } n = 1, 2, \ldots.$$

If $f(x)$ is odd $(f(-x) = -f(x))$ then $a_n = 0$. Similarly, if $f(x)$ is even $(f(-x) = f(x))$ then $b_n = 0$.

2. **Complex Form.**

   Fourier series can be expressed neatly in complex form as follows

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left[\frac{a_n}{2}(e^{in\pi x/L} + e^{-in\pi x/L}) + \frac{b_n}{2i}(e^{in\pi x/L} - e^{-in\pi x/L})\right].$$

   If we define

$$c_0 = \frac{a_0}{2}, \; c_n = \frac{a_n - ib_n}{2}, \; c_{-n} = \frac{a_n + ib_n}{2},$$

   where c_{-n} is the complex conjugate of c_n, i.e. $c_{-n} = c_n^*$, then $f(x)$ can be written as

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{in\pi x/L}, \tag{2.6}$$

   where the coefficients c_n can be determined from the formulas of a_n and b_n as

$$c_n = \frac{1}{2L}\int_0^{2L} f(x)e^{-in\pi x/L}\,dx. \tag{2.7}$$

   In many applications, particularly in analyzing of real situations, the function $f(x)$ to be approximated is known only on a discrete set of "sampling points" of x. Hence, the integral (2.7) cannot be evaluated in a closed form and Fourier analysis cannot be applied directly. It then becomes necessary to replace continuous Fourier analysis by a discrete version of it.

3. **Discrete Fourier Transform.**

   The linear *discrete Fourier transform* [84] of a periodic (discrete) sequence of complex values $u_0, \ldots, u_{N_{\mathcal{F}}-1}$ with period $N_{\mathcal{F}}$, is a sequence of periodic complex values $\hat{u}_0, \ldots, \hat{u}_{N_{\mathcal{F}}-1}$ defined by

$$\hat{u}_k = \frac{1}{N_{\mathcal{F}}}\sum_{j=0}^{N_{\mathcal{F}}-1} u_j e^{-2\pi ijk/N_{\mathcal{F}}}, \qquad k = 0, 1, \ldots, N_{\mathcal{F}} - 1. \tag{2.8}$$

   The linear inverse transformation is

$$u_j = \sum_{k=0}^{N_{\mathcal{F}}-1} \hat{u}_k e^{2\pi ijk/N_{\mathcal{F}}}, \qquad j = 0, 1, \ldots, N_{\mathcal{F}} - 1. \tag{2.9}$$

   The most obvious application of discrete Fourier analysis consists in the numerical calculation of Fourier coefficients. Suppose we want to approximate a real-valued periodic function $f(x)$, defined on the interval $[0, 2L]$ that is sampled with an even number $N_{\mathcal{F}}$ of grid points

$$x_j = jh, \; h = 2L/N_{\mathcal{F}}, \; j = 0, 1, \dots, N_{\mathcal{F}} - 1,$$

by its Fourier series (2.6). First we compute approximate values of the Fourier coefficients c_n (2.7) by the discrete Fourier transform (2.8)

$$\hat{c}_k \approx \frac{1}{N_{\mathcal{F}}} \sum_{j=0}^{N_{\mathcal{F}}-1} f(x_j) e^{-2\pi ijk/N_{\mathcal{F}}}, \tag{2.10}$$

and then truncate the Fourier series (2.6) formed with these approximate coefficients. Because the discrete Fourier transform and its inverse exhibit periodicity with period $N_{\mathcal{F}}$, i.e. $\hat{u}_{k+N_{\mathcal{F}}} = \hat{u}_k$ (this property results from the periodic nature of $e^{2\pi ijk/N_{\mathcal{F}}}$), it makes no sense to use more than $N_{\mathcal{F}}$ terms in the series, and it suffices to calculate one full period. Thus, the range of Fourier modes distinguishable on the grid, discretized with an even number $N_{\mathcal{F}}$ of grid points, is $k = -N_{\mathcal{F}}/2 + 1, \dots, N_{\mathcal{F}}/2$ and the Fourier series (2.6) formed with the approximate coefficients (2.10) is

$$\hat{f}(x) \approx \sum_{k=-N_{\mathcal{F}}/2+1}^{N_{\mathcal{F}}/2} \hat{c}_k e^{ik\pi x/L}. \tag{2.11}$$

The function $\hat{f}(x)$ not only approximates, but actually interpolates $f(x)$ at the sampling grid points x_j.

The approximation of large amounts of equally spaced data by trigonometric polynomials can produce very accurate results. Fourier's theorem states that at a point where the function $f(x)$ is continuous, the Fourier series converges to the value of $f(x)$, with a speed related to the smoothness of the function. The smoother the function $f(x)$ is, the more rapidly the series converges. The subsequent rapid decay of the coefficients implies that the Fourier series can be truncated after a few terms.

For discontinuous functions with bounded variation, at the point of discontinuity the Fourier series converges to the average of the values on either side of the discontinuity, and the rate of the coefficients' decay is $O(1/n)$; and if $f(x)$ is continuous but the first derivative $df(x)/dx$ is discontinuous, then the rate is $O(1/n^2)$ and so on.

4. **Matrix Form.**

   In matrix form, the discrete Fourier transform (2.8) can be written as

$$\hat{u}_k = \frac{1}{N_{\mathcal{F}}} M_{kj} u_j, \qquad k, j = 0, 1, \ldots, N_{\mathcal{F}} - 1, \tag{2.12}$$

   where $M_{kj} = \omega^{kj}$ and $\omega = e^{-2\pi i/N_{\mathcal{F}}}$ is the $N_{\mathcal{F}}$th root of unity, so

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \ldots & \omega^{N_{\mathcal{F}}-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \ldots & \omega^{2(N_{\mathcal{F}}-1)} \\ \vdots & \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & \omega^{N_{\mathcal{F}}-1} & \omega^{2(N_{\mathcal{F}}-1)} & \omega^{3(N_{\mathcal{F}}-1)} & \ldots & \omega^{(N_{\mathcal{F}}-1)(N_{\mathcal{F}}-1)} \end{pmatrix}.$$

   Similarly, the inverse discrete Fourier transform (2.9) has the form

$$u_j = M^*_{kj} \hat{u}_k, \qquad k, j = 0, 1, \ldots, N_{\mathcal{F}} - 1, \tag{2.13}$$

   where $M^*_{kj} = (\omega^*)^{kj}$ and ω^* is the complex conjugate of ω.

In the early years, the impact of discrete Fourier analysis was limited by the very large number of arithmetic calculations required by the theory in its naive form (number of multiplications required is $O(N^2_{\mathcal{F}})$). This was changed in 1965 by the invention of the mathematical algorithm of **Cooley** and **Tukey** [18], which has become known as the "Fast Fourier Transform" (FFT).

The FFT algorithm reduces the computational work required to carry out a discrete Fourier transform by reducing the number of multiplications and additions of (2.13) (computational time is reduced from $O(N^2_{\mathcal{F}})$ to $O(N_{\mathcal{F}} \log N_{\mathcal{F}})$). This algorithm is useful in situations where the number of grid points can be chosen to be a highly composite number. Since 1965, the FFT usage has expanded and led to a revolution in the use of trigonometric polynomial approximations.

2.3.2 Numerical Derivatives

To apply spectral methods to a partial differential equation we need to evaluate derivatives of functions. Suppose that we have a periodic real-valued function $f(x)$ with period $2L$, defined on the interval $[0, 2L]$ that is discretized with an even number $N_{\mathcal{F}}$ of grid points, so that the grid size $h = 2L/N_{\mathcal{F}}$. The complex form of the Fourier series representation of $f(x)$ is

$$\hat{f}(x) \approx \sum_{k=-N_{\mathcal{F}}/2+1}^{N_{\mathcal{F}}/2} \hat{c}_k e^{ik\pi x/L}. \tag{2.14}$$

At $k = N_{\mathcal{F}}/2$, the above series (2.14) gives a term $\hat{c}_{N_{\mathcal{F}}/2}e^{iN\pi x/(2L)}$, which alternates between $\pm\hat{c}_{N_{\mathcal{F}}/2}$ at the grid point $x_j = jh,\ j = 0, 1, \ldots, N_{\mathcal{F}} - 1$, and since it cannot be differentiated, we should set its derivative to be zero at the grid points.

The numerical derivatives of the function $f(x)$ can be illustrated as a matrix multiplication. For the first derivative, we multiply the Fourier coefficients (2.10) by the corresponding differentiation matrix

$$D_1 = \text{Diag}\left(0, 1, 2, 3, \ldots, \frac{N_{\mathcal{F}}}{2} - 1, 0, -\left(\frac{N_{\mathcal{F}}}{2} - 1\right), \ldots, -3, -2, -1\right)\frac{i\pi}{L},$$

for an even number $N_{\mathcal{F}}$ of grid points. This matrix has non-zero elements only on the diagonal. For an odd number $N_{\mathcal{F}}$ of grid points, the differentiation matrix corresponding to the first derivative is diagonal with elements

$$(0, 1, 2, \ldots, (N_{\mathcal{F}} - 1)/2, -(N_{\mathcal{F}} - 1)/2, \ldots, -1)i\pi/L.$$

Then, we compute an inverse discrete Fourier transform using (2.11) to return to the physical space and deduce the first derivative of $f(x)$ on the grid.

Similarly, taking the second derivative corresponds to the multiplication of the Fourier coefficients (2.10) by the corresponding differentiation matrix D_2 which is diagonal with elements

$$-\left(0, 1, 4, \ldots, \left(\frac{N_{\mathcal{F}}}{2} - 1\right)^2, \left(\frac{N_{\mathcal{F}}}{2}\right)^2, \left(\frac{N_{\mathcal{F}}}{2} - 1\right)^2, \ldots, 4, 1\right)\frac{\pi^2}{L^2},$$

for an even number $N_{\mathcal{F}}$ of grid points.

In general, in case of an even number $N_{\mathcal{F}}$ of grid points, approximating the mth numerical derivatives of a grid function $f(x)$ corresponds to the multiplication of the Fourier coefficients (2.10) by the corresponding differentiation matrix which is diagonal with elements $((ik\pi/L)^m)$ for

$$k = 0, 1, 2, 3, \ldots, \frac{N_{\mathcal{F}}}{2} - 1, \frac{N_{\mathcal{F}}}{2}, -\left(\frac{N_{\mathcal{F}}}{2} - 1\right), \ldots, -3, -2, -1,$$

with the exception that for odd derivatives we set the derivative of the highest mode $k = N_{\mathcal{F}}/2$ to be zero.

2.3.3 An Example

Discrete Fourier transforms (2.10) are often used to solve partial differential equations. The exponential basis functions are eigenfunctions of differentiation, which means that this representation transforms any linear differential equation with constant coefficients into ordinary differential equations. One then uses the inverse DFT (2.11) to transform the result back into the ordinary spatial representation. Such an approach is called a spectral method.

Consider again the linear diffusion equation

$$\frac{\partial u(x,t)}{\partial t} = \nu \frac{\partial^2 u(x,t)}{\partial x^2}, \qquad t_0 \le t \le T,\ 0 \le x \le 2L, \tag{2.15}$$

subject to periodic boundary conditions and an initial condition at $t_0 = 0$

$$u(x, t = 0) = u_0(x).$$

Suppose that the space interval $[0, 2L]$ is discretized with an even number $N_{\mathcal{F}}$ of grid points and the solution $u(x,t)$ is represented by its Fourier series as follows

$$u(x,t) = \frac{a_0(t)}{2} + \sum_{n=1}^{N_{\mathcal{F}}/2} a_n(t) \cos(n\pi x/L) + \sum_{n=1}^{N_{\mathcal{F}}/2-1} b_n(t) \sin(n\pi x/L). \tag{2.16}$$

Note that the Fourier series (2.16) satisfies the boundary conditions of the problem, i.e. $u(0,t) = u(2L,t),\ u(x,t) = u(x+2L,t)$.

Differentiating (2.16) with respect to t once, and with respect to x twice yields

$$\frac{\partial u(x,t)}{\partial t} = \frac{1}{2}\frac{da_0(t)}{dt} + \sum_{n=1}^{N_{\mathcal{F}}/2} \frac{da_n(t)}{dt} \cos(n\pi x/L) + \sum_{n=1}^{N_{\mathcal{F}}/2-1} \frac{db_n(t)}{dt} \sin(n\pi x/L), \tag{2.17}$$

and

$$\frac{\partial^2 u(x,t)}{\partial x^2} = -\sum_{n=1}^{N_{\mathcal{F}}/2} a_n(t)(n\pi/L)^2 \cos(n\pi x/L) - \sum_{n=1}^{N_{\mathcal{F}}/2-1} b_n(t)(n\pi/L)^2 \sin(n\pi x/L), \tag{2.18}$$

respectively. Substituting (2.17) and (2.18) into the diffusion equation (2.15), and equating for the Fourier coefficients reduces the PDE to an uncoupled system of ODEs

$$\begin{aligned} \frac{da_0(t)}{dt} &= 0, \\ \frac{da_n(t)}{dt} &= -\nu(n\pi/L)^2 a_n(t), \\ \frac{db_n(t)}{dt} &= -\nu(n\pi/L)^2 b_n(t). \end{aligned}$$

This system can be solved analytically, i.e. $a_n(t) = a_n(0)\exp(-\nu(n\pi/L)^2 t)$, etc., where $a_n(0)$ are the Fourier coefficients of the Fourier series representation of the initial condition $u_0(x)$, and so no numerical solution is needed.

For nonlinear PDEs, the nonlinear terms are evaluated by transforming from spectral space to physical space to find the values of these terms at the grid points. Then one transforms back to Fourier space to work out derivatives. The resulting system of ODEs is coupled through the nonlinear terms, while the linear part is represented by a diagonal matrix in the Fourier basis. In this case, the system is not trivial to solve analytically and a numerical method is needed.

Chapter 3

Exponential Time Differencing (ETD) Methods

Outline of Chapter

The basic idea of the method of lines is to replace the spatial derivatives in a partial differential equation (PDE) with algebraic approximations. Thus, we have a coupled system of ODEs with only time remaining as an independent variable. Now we can apply any existing well established numerical methods to compute an approximate numerical solution to the PDE.

Exponential Time Differencing (ETD) schemes are time integration methods that can be efficiently combined with spatial approximations to provide accurate smooth solutions for stiff or highly oscillatory semi-linear PDEs. The work reported in this chapter gives the derivation of the explicit ETD schemes for arbitrary order following the approach in [9, 17, 19, 63], and presents the explicit Runge-Kutta (ETD-RK) versions of these schemes constructed by **Cox** and **Matthews** [19].

In addition, the work contains an analytical examination of the methods' stability properties, which determines the range of time step for which the method is numerically stable. The approach computes the boundaries of the stability regions for a general test problem for the explicit ETD methods of multi-step or RK type up to fourth-order. The stability regions are illustrated in two-dimensional plots for various negative and purely real stiff parameters of the test problem. The plots demonstrate the ability of these methods to use large time-step sizes. This gives them an advantage over the ordinary explicit time-discretization methods which have severe restrictions on the selection of time step size for reason of stability when solving stiff problems.

3.1 Introduction

Many physical phenomena can be represented by partial differential equations (PDEs). When discretizing the spatial part of these equations (see §2), one commonly obtains a stiff system of coupled ordinary differential equations (ODEs) in time t. Stiff systems are routinely encountered in scientific applications and are characterized by having a large range of time scales. Often the large-scale solution sought varies much more slowly in time than small scales that decay or disperse rapidly, or have both features of rapid decay and rapid oscillation. In other words, stiff problems arise in areas where vastly different time scales all play a role in the overall solution of the PDE.

Stiffness can also be inherent in the problem due to the wide range of the eigenvalues, (i.e. the eigenvalues differ greatly in magnitude), of the differentiation matrix applied to discretize the spatial derivatives in a PDE (see §2.2.3 and §2.3.2). These eigenvalues spread out and become even larger as we increase the number of points with which we are discretizing the spatial operator. The stiffness problem is also exacerbated when a PDE has higher order spatial derivatives than the second. For such problems, numerical integrators require particular handling to achieve a precise solution to the problem.

As mentioned in §1, applying conventional explicit time stepping schemes to a stiff system requires the least number of computations per time step, but the stability requirement restricts the size of the time step to be very small to resolve the transient (rapidly-varying) part of the solution. Implicit time stepping schemes have much better stability properties compared to conventional explicit integrators, and allow significantly larger time steps that do not introduce instabilities. However, the number of computations required to solve a large nonlinear system of ODEs at each time step increases significantly.

Numerous time discretization methods that are designed to handle stiff systems have been developed. One example is the family of **Exponential Time Differencing** (ETD) schemes. This class of schemes is especially suited to semi-linear problems which can be split into a linear part, which contains the stiffest part of the dynamics of the problem, and a nonlinear part, which varies more slowly than the linear part. These schemes have been rediscovered several times in various forms and under various names [15, 17, 37, 49, 53, 61, 63, 82]. An example is the **Exact Linear Part (ELP)** schemes that were derived in [9] for arbitrary order. However, the authors of [9] did not give explicit formulas for the methods' coefficients. In a subsequent paper, **Cox** and **Matthews** [19] gave an explicit derivation of the explicit ELP methods, for arbitrary order s, with explicit formulas for the methods' coefficients and referred to these methods as the Exponential Time Differencing (ETD) schemes (the term used arose originally in the field of computational electrodynamics [40, 65, 71]). In addition, the authors of [19] further constructed new explicit **Runge-Kutta** (ETD-RK) versions of these schemes up to fourth-order.

In §3.2, we follow the approach in [9, 17, 19, 63] and present the algorithm derivation for the explicit ETD schemes. In the first step, the ETD schemes recover the exact solution to the linear part, which (numerically) is the difficult part (stiff or oscillatory in nature) of the differential equation, in a similar way to the approach of the **Integrating Factor (IF)** schemes [7, 8, 11, 19, 44, 45, 52, 57, 84] (see §3.2.1 for further details concerning the approach of the IF methods). The next step in the derivation is to integrate exactly an approximation of the nonlinear terms. We may approximate the nonlinear parts by some polynomial in time t that may be calculated using previous steps of the integration process, thus producing multi-step ETD methods (see §3.2.2) or by RK-like stages, resulting in ETD schemes of Runge-Kutta type (see §3.2.3). The coefficients of the ETD methods are the exponential and related functions of the linear operators. These coefficients can be evaluated once before the integration begins if a constant time step is used throughout the integration, see §4 for further details.

The convergence analysis for the explicit s-step exponential schemes was carried out in [7, 15, 57] for solving semi-linear equations. The analysis showed that the schemes achieve order of accuracy s, for appropriate starting values at the nth and previous time steps. In addition, the authors of [37, 39] analyzed the convergence behavior of the explicit exponential Runge-Kutta methods for integrating semi-linear parabolic problems. They gave a new derivation of the classical order conditions and showed convergence for these methods up to order four.

In §3.3, we illustrate some key features of the explicit ETD schemes such as their stability properties. We follow the approach developed in [9, 19] for constructing stability regions of the ETD (in §3.3.1) and the ETD-RK (in §3.3.2) schemes of orders up to four. The stability regions are plotted in two dimensions, considering the case where the stiffness parameter in a general test problem is negative and purely real. We analyze these plots in order to show that these methods are capable of avoiding the rigorous ceiling imposed on the selection of the time step size when solving stiff problems with conventional explicit time discretization methods. The overall results are summarized in §3.4.

3.2 Algorithm Derivation

We begin by giving briefly the main idea behind the Lawson Integrating Factor IF methods [52], then we give, in detail, the algorithm derivation for the explicit ETD schemes [9, 17, 19, 63].

Consider stiff semi-linear PDEs that can be written in the form

$$\frac{\partial u(x,t)}{\partial t} = \mathcal{L}u(x,t) + \mathcal{F}(u(x,t),t), \tag{3.1}$$

where the linear operator $\mathcal{L}$ contains higher-order spatial derivatives than those contained in the nonlinear operator $\mathcal{F}$, and is mainly the term responsible for stiffness. For problems with spatially periodic boundary conditions, we use Fourier spectral methods [25, 83, 84] to discretize the spatial derivatives of (3.1) (see §2 for more details), and hence obtain a stiff system of coupled ODEs in time t

$$\frac{du(t)}{dt} = \text{Ł}u(t) + F(u(t),t). \tag{3.2}$$

The linear part Ł of the system is represented by a diagonal matrix, and F represents the action of the nonlinear operator on u on the grid. For problems where the boundary conditions are not periodic, we use finite difference formulas [58, 83] or Chebyshev polynomials [11, 25, 83, 84], and in this case, the linearized system is represented by a non-diagonal matrix. For dissipative PDEs, the eigenvalues of the matrix Ł are negative and real, whereas they are imaginary for dispersive PDEs. Dissipation in a dynamical system represents the concept of important mechanical modes, such as waves or oscillations, losing energy over time. Such systems are called dissipative systems. On the other hand, a dispersive PDE represents a system in which waves of different frequencies propagate at different phase velocities (the phase velocity is the rate at which the phase of the wave propagates in space).

For the stiff system of ODEs (3.2), the eigenvalues of the matrix Ł vary widely in magnitude, and the stiffness is caused by the eigenvalues of large magnitude. A competitive time stepping method should be able to integrate the system (3.2) accurately without requiring very small time steps for the largest magnitude eigenvalue. Simultaneously it should be able to handle small eigenvalues. The nonlinear term F

requires an explicit treatment since fully implicit methods are too costly for a large system of ODEs.

To derive the time discretization methods (IF and ETD methods), we consider for simplicity a single model of a stiff ODE

$$\frac{du(t)}{dt} = cu(t) + F(u(t), t), \tag{3.3}$$

where the stiffness parameter c is either large, negative and real, or large and imaginary, or complex with large, negative real part and $F(u(t), t)$ is the nonlinear forcing term.

3.2.1 Integrating Factor Methods

The main idea behind the IF schemes is to use a change of variables

$$w(t) = u(t)e^{-ct},$$

so that when differentiating both sides of this equation we obtain

$$\frac{dw(t)}{dt} = \Big(\frac{du(t)}{dt} - cu(t)\Big)e^{-ct},$$

and then substituting from equation (3.3) we get

$$\begin{aligned} \frac{dw(t)}{dt} &= F(u(t), t)e^{-ct}, \\ &= F(w(t)e^{ct}, t)e^{-ct}. \end{aligned} \tag{3.4}$$

The aim now is to use any numerical integrator (IF schemes can be generalized to arbitrary order by applying any multi-step or Runge-Kutta methods) on the transformed nonlinear differential equation (3.4). The approximated solution is then transformed back to provide an approximate solution for the original u variable. For example, we can choose to apply the Euler method [14] to the transformed differential equation (3.4) as follows

$$w_{n+1} = w_n + \Delta t F(w_n e^{ct_n}, t_n)e^{-ct_n},$$

where Δt is the time step size and w_n denotes the numerical approximation to $w(t_n)$, and then transform back to the original variable to obtain the solution approximation. This yields the first-order **Integrating Factor Euler (IFEULER)** method [11, 84]

$$u_{n+1} = (u_n + \Delta t F_n)e^{c\Delta t}, \tag{3.5}$$

where u_n and F_n denote the numerical approximation to $u(t_n)$ and $F(u(t_n), t_n)$ respectively.

The purpose of transforming the differential equation (3.3) to equation (3.4), is to remove the explicit dependence in the differential equation on the operator c, except inside the exponential. Now the problem is no longer stiff since the linear "stiff" term of the differential equation (3.3), that constrains the stability, is gone. Therefore, it can be solved exactly with the possibility of larger time steps. However, for PDEs with slowly varying nonlinear terms, the introduction of the fast decay time scale into the nonlinear term introduces large errors [7, 11, 19, 49] into the system.

3.2.2 Exponential Time Differencing Methods

To derive the s-step ETD schemes (the derivation is taken from [9, 17, 19, 63]), we follow an approach similar to that of deriving the IF schemes, i.e. we multiply (3.3) through by the integrating factor e^{-ct}, and then integrate the equation over a single time step from $t = t_n$ to $t = t_{n+1} = t_n + \Delta t$ to get

$$u(t_{n+1}) = u(t_n)e^{c\Delta t} + e^{c\Delta t}\int_0^{\Delta t} e^{-c\tau}F(u(t_n+\tau), t_n+\tau)d\tau. \tag{3.6}$$

This formula is *exact*, and the next step is to derive approximations to the integral in equation (3.6). This procedure does not introduce an unwanted fast time scale into the solution and the schemes can be generalized to arbitrary order.

If we apply the Newton Backward Difference Formula [14], using information about $F(u(t), t)$ at the nth and previous time steps, we can write a polynomial approximation to $F(u(t_n+\tau), t_n+\tau)$ in the form

$$F(u(t_n+\tau), t_n+\tau) \approx G_n(t_n+\tau) = \sum_{m=0}^{s-1}(-1)^m\binom{-\tau/\Delta t}{m}\nabla^m G_n(t_n), \tag{3.7}$$

where ∇ is the backward difference operator defined as follows

$$\begin{aligned}\nabla^m G_n(t_n) &= \sum_{k=0}^{m}(-1)^k\binom{m}{k}G_{n-k}(t_{n-k}),\\ &\approx \sum_{k=0}^{m}(-1)^k\binom{m}{k}F(u(t_{n-k}), t_{n-k}),\end{aligned} \tag{3.8}$$

and

$$m!\binom{-\Lambda}{m} = (-\Lambda)(-\Lambda-1)\cdots(-\Lambda-m+1),\ m = 1, \ldots, s-1.$$

(note that $0!\binom{-\Lambda}{0} = 1$). If we substitute the approximation (3.7) in the integrand (3.6), we get

$$u(t_{n+1}) - u(t_n)e^{c\Delta t} \approx \Delta t \sum_{m=0}^{s-1} (-1)^m \int_0^1 e^{c\Delta t(1-\Lambda)} \binom{-\Lambda}{m} d\Lambda \nabla^m G_n(t_n), \tag{3.9}$$

where $\Lambda = \tau/\Delta t$.

We will indicate the integral in (3.9) by

$$\mathrm{g}_m = (-1)^m \int_0^1 e^{c\Delta t(1-\Lambda)} \binom{-\Lambda}{m} d\Lambda, \tag{3.10}$$

and then calculate the g_m by bringing in the generating function. For $z \in \mathbb{R}$, $|z| < 1$, we define the generating function

$$\begin{aligned} \Gamma(z) &= \sum_{m=0}^{\infty} \mathrm{g}_m z^m, \\ &= \int_0^1 e^{c\Delta t(1-\Lambda)} \sum_{m=0}^{\infty} \binom{-\Lambda}{m} (-z)^m d\Lambda, \\ &= \int_0^1 e^{c\Delta t(1-\Lambda)} (1-z)^{-\Lambda} d\Lambda, \\ &= \frac{e^{c\Delta t}(1 - z - e^{-c\Delta t})}{(1-z)(c\Delta t + \log(1-z))}. \end{aligned} \tag{3.11}$$

Rearranging (3.11) to the form

$$(c\Delta t + \log(1-z))\Gamma(z) = e^{c\Delta t} - (1-z)^{-1},$$

and expanding as a power series in z

$$\left(c\Delta t - z - \frac{z^2}{2} - \frac{z^3}{3} - \cdots\right)(\mathrm{g}_0 + \mathrm{g}_1 z + \mathrm{g}_2 z^2 + \cdots) = e^{c\Delta t} - 1 - z - z^2 - z^3 - \cdots,$$

we can find a recurrence relation for the g_m for $m \geq 0$ by equating like powers of z

$$\begin{aligned} c\Delta t \mathrm{g}_0 &= e^{c\Delta t} - 1, \\ c\Delta t \mathrm{g}_{m+1} + 1 &= \mathrm{g}_m + \tfrac{1}{2}\mathrm{g}_{m-1} + \tfrac{1}{3}\mathrm{g}_{m-2} + \cdots + \tfrac{1}{m+1}\mathrm{g}_0 = \textstyle\sum_{k=0}^{m} \frac{1}{m+1-k}\, \mathrm{g}_k. \end{aligned} \tag{3.12}$$

Having determined the g_m, the ETD schemes (3.9) then can be given in explicit forms.

Substituting (3.8) and (3.10) in (3.9), we deduce the general generating formula of the ETD schemes of order s [19]

$$u_{n+1} = u_n e^{c\Delta t} + \Delta t \sum_{m=0}^{s-1} \mathrm{g}_m \sum_{k=0}^{m} (-1)^k \binom{m}{k} F_{n-k}, \tag{3.13}$$

where u_n and F_n denote the numerical approximation to $u(t_n)$ and $F(u(t_n), t_n)$ respectively, and the g_m are given by (3.12).

ETD Schemes

The first-order **ETD1** scheme [9, 15, 19, 61]

$$u_{n+1} = u_n e^{c\Delta t} + (e^{c\Delta t} - 1)F_n/c, \tag{3.14}$$

is obtained by setting $s = 1$ in the explicit generating formula (3.13). Setting $s = 2$ in (3.13) gives us the second-order **ETD2** scheme [19]

$$u_{n+1} = u_n e^{c\Delta t} + \{((c\Delta t + 1)e^{c\Delta t} - 2c\Delta t - 1)F_n + (-e^{c\Delta t} + c\Delta t + 1)F_{n-1}\}/(c^2\Delta t). \tag{3.15}$$

If $s = 3$ in (3.13), we obtain the third-order **ETD3** scheme

$$\begin{aligned} u_{n+1} = u_n e^{c\Delta t} + \{&((2c^2\Delta t^2 + 3c\Delta t + 2)e^{c\Delta t} - 6c^2\Delta t^2 - 5c\Delta t - 2)F_n \\ &+(-(4c\Delta t + 4)e^{c\Delta t} + 6c^2\Delta t^2 + 8c\Delta t + 4)F_{n-1} \\ &+((c\Delta t + 2)e^{c\Delta t} - 2c^2\Delta t^2 - 3c\Delta t - 2)F_{n-2}\}/(2c^3\Delta t^2). \end{aligned} \tag{3.16}$$

Set $s = 4$ in (3.13) to achieve the fourth-order **ETD4** scheme

$$u_{n+1} = u_n e^{c\Delta t} + (\Phi_1 F_n - \Phi_2 F_{n-1} + \Phi_3 F_{n-2} - \Phi_4 F_{n-3})/(6c^4\Delta t^3), \tag{3.17}$$

where

$$\begin{aligned} \Phi_1 &= (6c^3\Delta t^3 + 11c^2\Delta t^2 + 12c\Delta t + 6)e^{c\Delta t} - 24c^3\Delta t^3 - 26c^2\Delta t^2 - 18c\Delta t - 6, \\ \Phi_2 &= (18c^2\Delta t^2 + 30c\Delta t + 18)e^{c\Delta t} - 36c^3\Delta t^3 - 57c^2\Delta t^2 - 48c\Delta t - 18, \\ \Phi_3 &= (6c^2\Delta t^2 + 24c\Delta t + 18)e^{c\Delta t} - 24c^3\Delta t^3 - 42c^2\Delta t^2 - 42c\Delta t - 18, \\ \Phi_4 &= (2c^2\Delta t^2 + 6c\Delta t + 6)e^{c\Delta t} - 6c^3\Delta t^3 - 11c^2\Delta t^2 - 12c\Delta t - 6. \end{aligned}$$

Note that as $c \to 0$ in the coefficients of the s-order ETD methods, the methods reduce to the corresponding order of the **Adams-Bashforth** schemes [43, 84]. For example, if we expand the exponential function, using Taylor series, in the first-order ETD1 method (3.14) as follows

$$u_{n+1} = u_n\left(1 + c\Delta t + \frac{(c\Delta t)^2}{2} + \frac{(c\Delta t)^3}{3!} + \cdots\right) + F_n\left(\Delta t + \frac{c\Delta t^2}{2} + \frac{c^2\Delta t^3}{3!} + \cdots\right),$$

and then take the limit as $c \to 0$, while keeping terms of $O(\Delta t)$, we obtain

$$u_{n+1} = u_n + \Delta t(cu_n + F_n) = u_n + \Delta t du(t)/dt,$$

which corresponds to the forward Euler method. In fact, in the case of $c = 0$, the explicit formulas of the coefficients involve division by zero, and for very small values of $|c|$, the coefficients suffer from rounding errors due to the large amount of cancellation in the formulas. To tackle this problem we can use the Taylor series instead of using the explicit formula of the coefficients, see §4 for a detailed discussion of this issue.

3.2.3 Exponential Time Differencing Runge-Kutta Methods

Generally, for the one-step time-discretization methods and the Runge-Kutta (RK) methods, all the information required to start the integration is available. However, for the multi-step time-discretization methods this is not true. These methods require the evaluations of a certain number of starting values of the nonlinear term $F(u(t), t)$ at the nth and previous time steps to build the history required for the calculations. Therefore, it is desirable to construct ETD methods that are based on RK methods.

ETD Runge-Kutta Schemes

Cox and **Matthews** [19] and **Friedli** [28] constructed a second-order ETD Runge-Kutta method, analogous to the "improved Euler" method given in [78], as follows. Putting $s = 1$ in (3.13) gives the first step

$$a_n = u_n e^{c\Delta t} + (e^{c\Delta t} - 1)F_n/c. \tag{3.18}$$

The term a_n approximates the value of u at $t_n + \Delta t$. The next step is to approximate F in the interval $t_n \leq t \leq t_{n+1}$, with

$$F = F_n + (t - t_n)(F(a_n, t_n + \Delta t) - F_n)/\Delta t + O(\Delta t^2),$$

and substitute into (3.6) to give the **ETD2RK1** scheme

$$u_{n+1} = a_n + (e^{c\Delta t} - c\Delta t - 1)(F(a_n, t_n + \Delta t) - F_n)/(c^2 \Delta t). \tag{3.19}$$

In a similar way, we can also form an **ETD2RK2** scheme analogous to the "modified Euler" method [78]. The first step

$$a_n = u_n e^{c\Delta t/2} + (e^{c\Delta t/2} - 1)F_n/c,$$

is formed by taking half a step of (3.18); then use the approximation

$$F = F_n + \frac{(t - t_n)}{\Delta t/2}(F(a_n, t_n + \Delta t/2) - F_n) + O(\Delta t^2),$$

in the interval $[t_n, t_n + \Delta t]$ in (3.6) to deduce the **ETD2RK2** scheme

$$u_{n+1} = u_n e^{c\Delta t} + \{((c\Delta t - 2)e^{c\Delta t} + c\Delta t + 2)F_n + 2(e^{c\Delta t} - c\Delta t - 1)F(a_n, t_n + \Delta t/2)\}/c^2\Delta t. \tag{3.20}$$

In fact there is a one-parameter family of such **ETD2RK$\jmath$** schemes. For $\jmath \in \mathbb{R}^+$, one can start with any fraction $1/\jmath$ of Δt for the first step (3.18) which gives

$$a_n = u_n e^{c\Delta t/\jmath} + (e^{c\Delta t/\jmath} - 1)F_n/c.$$

The term a_n approximates the value of u at $t_n + \Delta t/\jmath$. Next use the approximation

$$F = F_n + \frac{(t - t_n)}{\Delta t/\jmath}(F(a_n, t_n + \Delta t/\jmath) - F_n) + O(\Delta t^2),$$

in the interval $[t_n, t_n + \Delta t]$ in (3.6) to deduce the general ETD2RK$\jmath$ schemes as follows

$$u_{n+1} = u_n e^{c\Delta t} + \{((c\Delta t - \jmath)e^{c\Delta t} + (\jmath - 1)c\Delta t + \jmath)F_n + \jmath(e^{c\Delta t} - c\Delta t - 1)F(a_n, t_n + \Delta t/\jmath)\}/(c^2\Delta t).$$

In a similar way, for different values of the fraction $1/\jmath$ there are infinitely many third-order and fourth-order ETD-RK schemes. For example, the third-order **ETD3RK** scheme [19] which is analogous to the classical third-order RK method [14] is given by

$$\begin{aligned}
a_n &= u_n e^{c\Delta t/2} + (e^{c\Delta t/2} - 1)F_n/c, \\
b_n &= u_n e^{c\Delta t} + (e^{c\Delta t} - 1)(2F(a_n, t_n + \Delta t/2) - F_n)/c, \\
u_{n+1} &= u_n e^{c\Delta t} + \{((c^2\Delta t^2 - 3c\Delta t + 4)e^{c\Delta t} - c\Delta t - 4)F_n \\
&\quad + 4((c\Delta t - 2)e^{c\Delta t} + c\Delta t + 2)F(a_n, t_n + \Delta t/2) \\
&\quad + ((-c\Delta t + 4)e^{c\Delta t} - c^2\Delta t^2 - 3c\Delta t - 4)F(b_n, t_n + \Delta t)\}/(c^3\Delta t^2).
\end{aligned} \tag{3.21}$$

The terms a_n and b_n approximate the values of u at $t_n + \Delta t/2$ and $t_n + \Delta t$ respectively. The formula (3.21) is the quadrature formula for (3.6) derived from quadratic interpolation through the points t_n, $t_n + \Delta t/2$ and $t_n + \Delta t$.

Introducing a further parameter, a fourth-order scheme **ETD4RK** (taken from [19]) is obtained as follows:

$$\begin{aligned}
a_n &= u_n e^{c\Delta t/2} + (e^{c\Delta t/2} - 1)F_n/c, \\
b_n &= u_n e^{c\Delta t/2} + (e^{c\Delta t/2} - 1)F(a_n, t_n + \Delta t/2)/c, \\
c_n &= a_n e^{c\Delta t/2} + (e^{c\Delta t/2} - 1)(2F(b_n, t_n + \Delta t/2) - F_n)/c, \\
u_{n+1} &= u_n e^{c\Delta t} + \{((c^2\Delta t^2 - 3c\Delta t + 4)e^{c\Delta t} - c\Delta t - 4)F_n \\
&\quad + 2((c\Delta t - 2)e^{c\Delta t} + c\Delta t + 2)(F(a_n, t_n + \Delta t/2) + F(b_n, t_n + \Delta t/2)) \\
&\quad + ((-c\Delta t + 4)e^{c\Delta t} - c^2\Delta t^2 - 3c\Delta t - 4)F(c_n, t_n + \Delta t)\}/(c^3\Delta t^2).
\end{aligned} \tag{3.22}$$

The terms a_n and b_n approximate the values of u at $t_n + \Delta t/2$ and the term c_n approximates the value of u at $t_n + \Delta t$. The formula (3.22) is the quadrature formula for (3.6) derived from quadratic interpolation through the points t_n, $t_n + \Delta t/2$ and $t_n + \Delta t$, using average values of F at a_n and b_n.

In general, the ETD4RK method (3.22) has classical order four, but Hochbruck and Ostermann [38] showed that this method suffers from an order reduction. This is due to not satisfying some of the stiff order conditions. These conditions were derived [38] for explicit exponential Runge-Kutta methods applied to stiff semi-linear parabolic problems with homogeneous Dirichlet boundary condition and under appropriate temporal smoothness of the exact solution. They also presented numerical experiments which show that the order reduction, predicted by their theory, may in fact arise in practical examples. In the worst case, this leads to an order reduction to order three for the Cox and Matthews method (3.22) [19] and gives order four for Krogstad's method (ETDRK4-B) [49]. However, for certain problems, such as the numerical experiments conducted by Kassam and Trefethen [44, 45] for solving various one-dimensional diffusion-type problems, and the numerical results obtained in §5 for solving some dissipative and dispersive PDEs, the fourth-order convergence of the ETD4RK method [19] is confirmed numerically.

Finally, we note that as $c \to 0$ in the coefficients of the s-order ETD-RK methods, the methods reduce to the corresponding order of the **Runge-Kutta** schemes.

3.3 Stability Analysis

The stability of a given method for solving a system of ODEs is a theoretical measure of the extent to which the method produces satisfactory approximations. Stability is related to the accuracy of the methods and refers to errors not growing in subsequent steps. Such methods are called numerically stable. The stability analysis determines the range of time step for which the method is numerically stable. The stability region is the subset of the complex plane consisting of those $\Delta t\lambda \in \mathbb{C}$ for which, with time step Δt, the numerical approximation produces bounded solutions when applied to the scalar linear model problem $du(t)/dt = \lambda u(t)$.

In general, the linear stability analysis of time discretization methods is valid for a linear autonomous system of ODEs, linearized about a fixed point. This analysis only gives an indicator as to how stable the numerical methods are. It cannot be directly applied to solutions of nonlinear time-dependent PDEs with large amplitude since convergence and stability are solution-dependent issues.

Beylkin et al. [9] studied the stability for a family of explicit and implicit ELP schemes, and showed that these schemes have significantly better stability properties when compared with known Implicit-Explicit schemes. In addition, **Krogstad** [49] analyzed the stability regions of various time integrating methods, including the fourth-order ETDRK4-B method and multi-step generalizations of the IF methods, all of which he proposed, and the ETD4RK method of Cox and Matthews [19]. He deduced that the ETDRK4-B method has the largest stability region. Cox and Matthews [19] also studied the stability properties of the second-order ETD type schemes, while in [23], the study was for the ETD-RK schemes of [19] of orders up to and including the fourth. All authors concluded that ETD type schemes maintain good stability properties and can be widely applicable to dissipative PDEs and nonlinear wave equations.

The approach developed in [9, 19] for the stability analysis of composite schemes, i.e. schemes that use different methods for the linear and nonlinear parts of the equation, computes the boundaries of the stability regions for a general test problem. That is, to analyze the stability of the ETD schemes, we linearize the autonomous ODE

$$\frac{dv(t)}{dt} = cv(t) + F(v(t)), \tag{3.23}$$

about a fixed point u_0 (so that $cu_0 + F(u_0) = 0$), to obtain

$$\frac{du(t)}{dt} = cu(t) + \lambda u(t), \tag{3.24}$$

where $u(t)$ is the perturbation to u_0 and

$$\lambda = \left.\frac{dF(u(t))}{du}\right|_{u(t)=u_0}.$$

In order to keep the fixed point u_0 stable, we require $Re(c + \lambda) < 0$ (note that the fixed points of the ETD methods are the same as those of the ODE (3.23), in contrast to the IF methods which do not preserve the fixed points for the ODE that they discretize [19]. It seems desirable for a numerical method to fulfill this property with respect to capturing as much of the dynamics of the system as possible).

If both c and λ are complex, the stability region is four-dimensional. But if both c and λ are purely imaginary [26] or purely real [19], or if λ is complex and c is fixed and real [9] then the stability region is two-dimensional.

Our study concentrates on two cases, where first, λ is complex and c is fixed, negative and purely real and second, c is negative and both c and λ are purely real. The stability regions are constructed for the ETD and the ETD-RK methods in §3.3.1 and §3.3.2 respectively.

3.3.1 Stability of Exponential Time Differencing Methods

When applying the first-order ETD1 method (3.14) to the linearized problem (3.24), we obtain

$$u_{n+1} = u_n e^{c\Delta t} + \lambda u_n (e^{c\Delta t} - 1)/c.$$

Defining $r = u_{n+1}/u_n,\ x = \lambda \Delta t$ and $y = c\Delta t$, leads to

$$r = e^y + \frac{x}{y}(e^y - 1). \tag{3.25}$$

If the second-order ETD2 method (3.15) is applied to (3.24), the linearization of the nonlinear term in the numerical method yields a recurrence relation involving u_{n+1}, u_n and u_{n-1}, and the equation for the factor r [19] by which the solution is multiplied after each step is

$$y^2 r^2 - (y^2 e^y + [(y+1)e^y - 2y - 1]x)r + (e^y - y - 1)x = 0. \tag{3.26}$$

In a similar way, when applying the ETD3 (3.16) and the ETD4 (3.17) schemes to (3.24), a recurrence relation is obtained from the linearization, and the equation for r is a third and fourth-order polynomial for the ETD3 and the ETD4 schemes respectively. The formulas of the factor r for the ETD3 and the ETD4 methods are not given explicitly here since they are very cumbersome.

We commence our analysis by choosing real negative values of the constant c, i.e. varying $y = c\Delta t$, and looking for a region of stability in the complex x plane where the solution u_n remains bounded as $n \to \infty$. The solution for $r = u_{n+1}/u_n$ can be sought in the form $r = r_1 e^{i\theta}$. Evidently, the solution decays if $r_1 < 1$. Hence, the boundary of the stability region is determined by writing $r = e^{i\theta},\ \theta \in [0, 2\pi]$ in each equation for the factor r for the ETD1 (3.14), the ETD2 (3.15), the ETD3 (3.16) and the ETD4 (3.17) methods and then by solving for $x = \lambda\Delta t$. The corresponding family of stability regions are plotted in the complex x plane and displayed in figures 3.1, 3.2 and 3.3. Note that, in these figures, the horizontal and the vertical axes represent real x (Re(x)) and imaginary x (Im(x)) respectively.

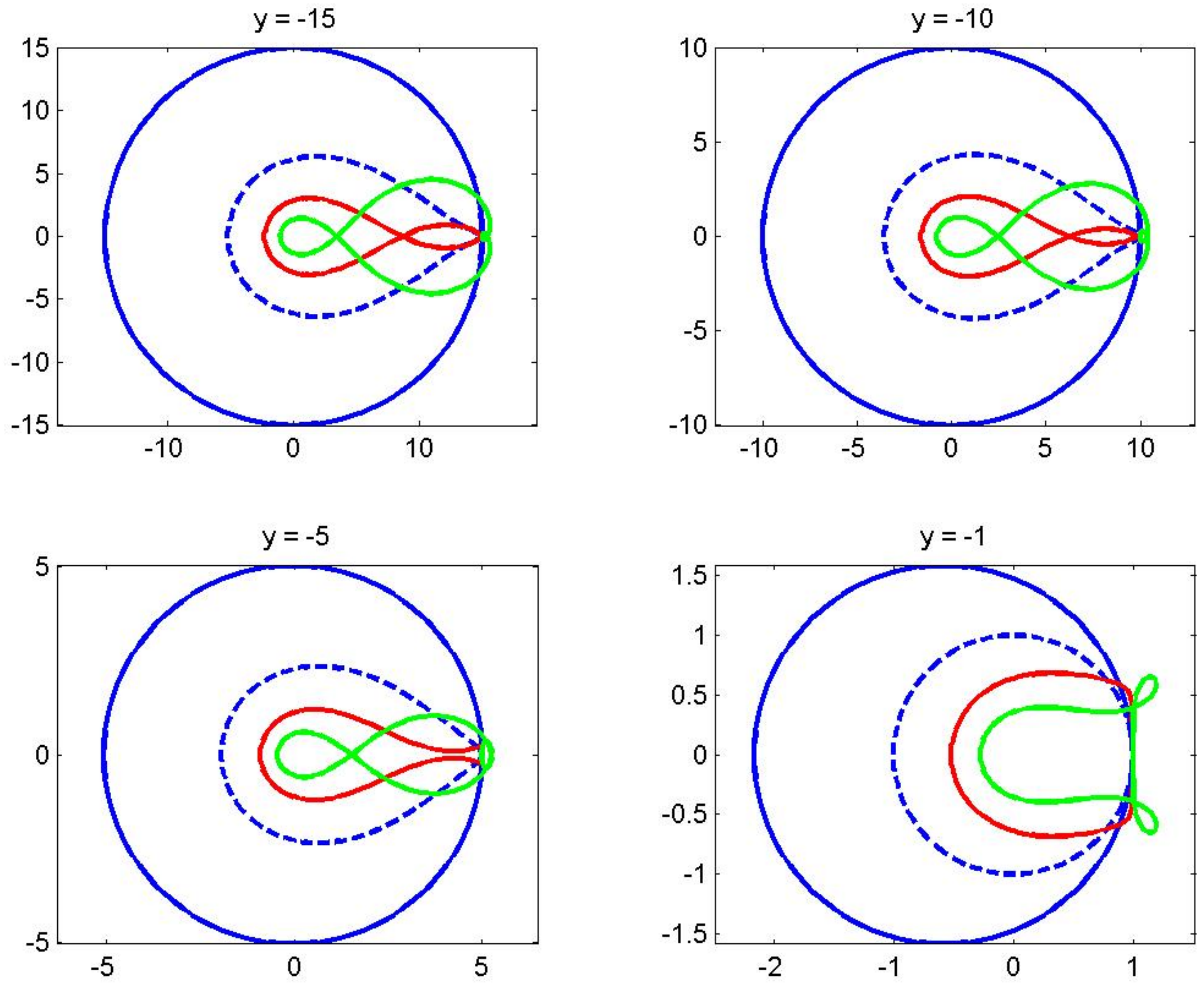


FIGURE 3.1: Stability regions in the complex x plane. The four methods are: ETD1 (blue-solid), ETD2 (blue-dashed), ETD3 (red-solid), ETD4 (green-solid).

In figure 3.1, notice first that, for each fixed value of $y = -1, -5, -10, -15$, the stability region of the ETD1 and the ETD2 schemes is the interior of the curves, which are simple and closed; while for the ETD3 and the ETD4 schemes, it is only the interior of those portions of the curves that contain the origin. Next, observe that the ETD1 scheme has the largest stability region while the ETD4 scheme has the smallest. In fact, as shown in figure 3.1, the stability region, for each fixed value of y, shrinks as the method's order increases.

Figures 3.2 and 3.3 illustrate the complex plot of the same four methods with different values of y. The outer curves for the ETD1, the ETD2 and the ETD3 schemes correspond to $y = -6$, and to $y = -3$ for the ETD4 scheme. The inner blue

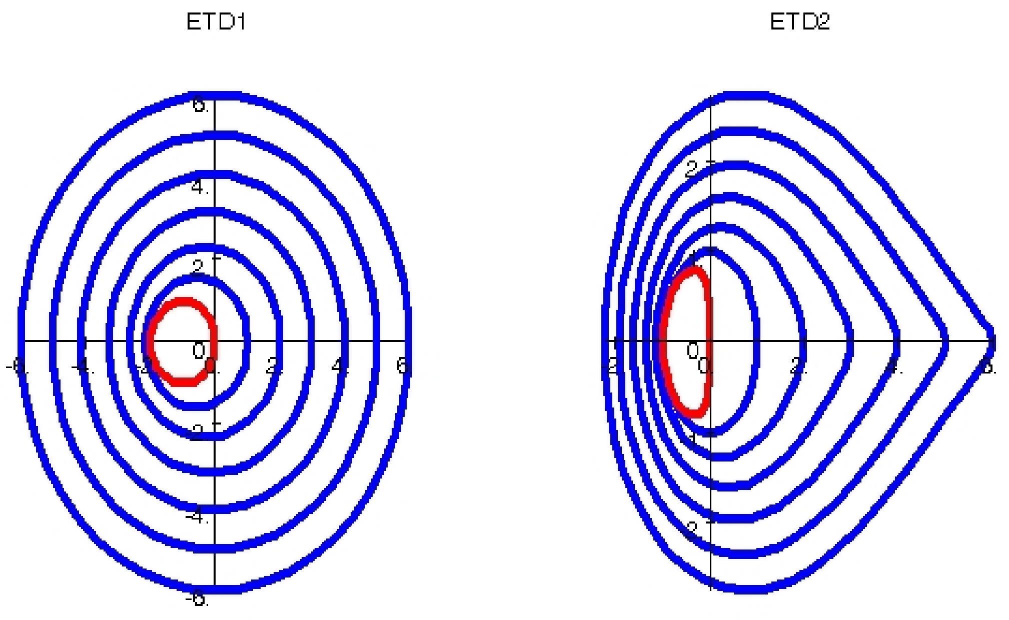


FIGURE 3.2: Stability regions in the complex x plane. The curves for the ETD1 and the ETD2 schemes correspond to $y = -6, -5, -4, -3, -2, -1$, from the outer curve to the inner curve respectively. The inner red curves correspond to $y = 0$.

curves for all four schemes correspond to $y = -1$. The inner red curves correspond to the case $y = 0$, where the stability regions coincide with those of the corresponding order **Adams-Bashforth** schemes [43, 84]. This is expected since in the limit $y \to 0$, ETD schemes turn into the corresponding order explicit **Adams-Bashforth** schemes [9]. In the limit $y \to -\infty$, we find that the stability region of each of the four methods preserves its shape and each grows larger, which allows the methods to use a large time-step size ($\Delta t = O(1)$ as $|c| \to \infty$) when solving stiff problems. On the contrary, the stability regions of the conventional explicit numerical methods preserve their size whatever the value of the stiffness parameter is (for example, the stability region of the Euler methods is a fixed circle of radius 1), which forces the methods to use very small time-step sizes ($\Delta t = O(1/|c|)$) when integrating stiff problems, see §5 for illustrative examples.

As shown in figure 3.2, the boundary of the stability region for the ETD1 and the ETD2 schemes passes through the point $x = -y$ (this is true for any fixed value of y), which agrees with the result found in [19] for the ETD2 schemes. This feature is consistent with the true stability boundary of the differential equation (3.24) of the

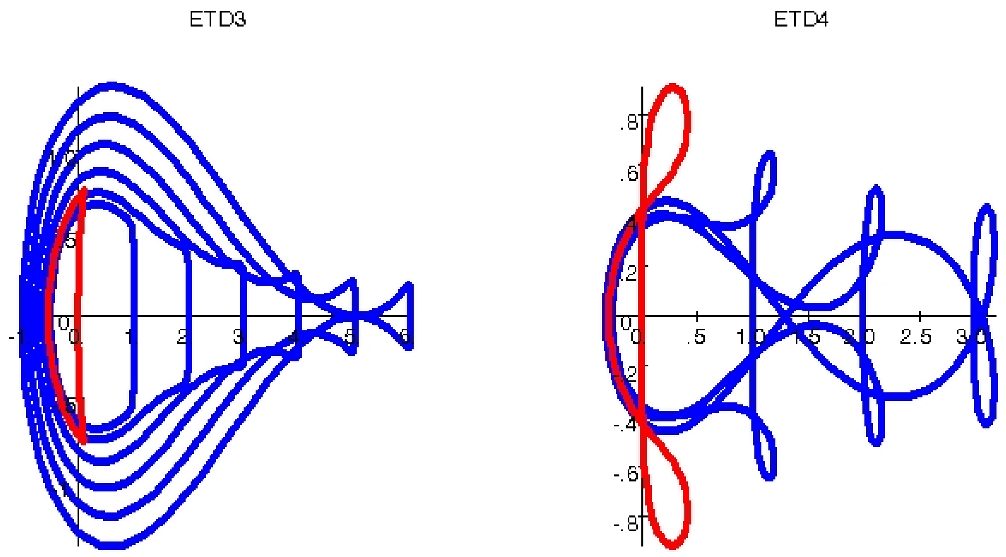


FIGURE 3.3: Stability regions in the complex x plane. The curves for the ETD3 scheme correspond to $y = -6, -5, -4, -3, -2, -1$, from the outer curve to the inner curve respectively. The same holds for the ETD4 scheme for $y = -3, -2, -1$. The inner red curves correspond to $y = 0$.

test problem, namely, the solution decays for $Re(c+\lambda) < 0$ and it grows otherwise.

For the ETD3 method, we find that for values of $y \in [-6, 0)$ the curves of its region do not cross and hence the stability region has a simple structure and passes through the point $x = -y$, see figure 3.3. As y decreases, the curves of the region cross over and the region develops to a more complicated structure, separating into several portions. The stability regions for the ETD3 method for values of $y \in [-10, -6)$ (not shown) are the interior of those portions of the curves that contain the origin, and the interior of those portions of the curves where the boundary passes through the point $x = -y$, whereas for values of $y = -10, \ldots, -\infty$ they are only the interior of those portions of the curves that contain the origin, see figure 3.1 for $y = -10, -15$.

For the ETD4 method, we find that the curves of its region cross and hence the region separates into several portions. For values of $y \in [-2, 0)$ the stability regions are the interior of those portions of the curves that contain the origin, see figure 3.3. For values of $y \in [-3, -2)$ (not shown) the stability regions are the interior of those portions of the curves that contain the origin, and the interior of those portions of

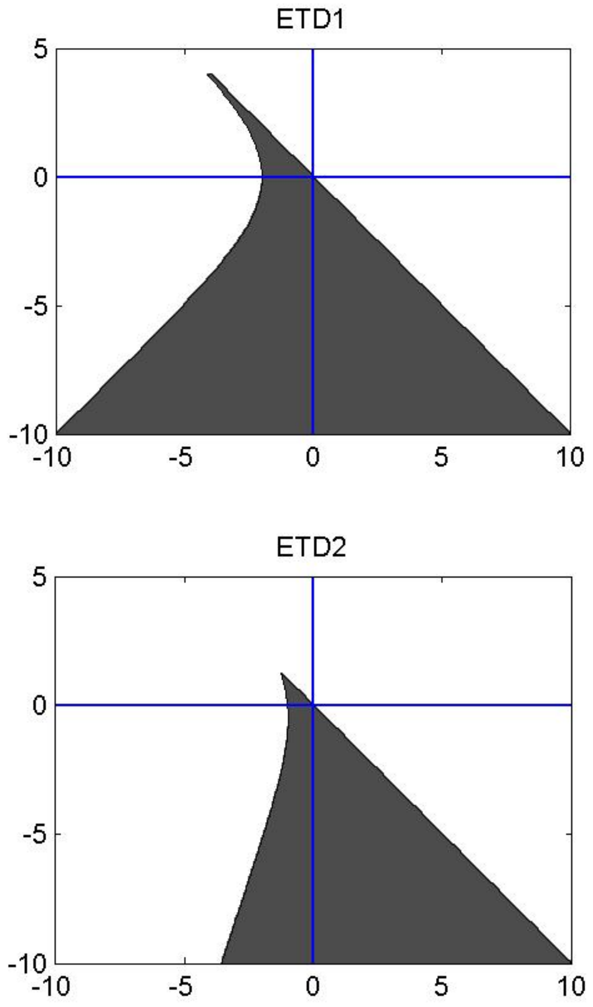
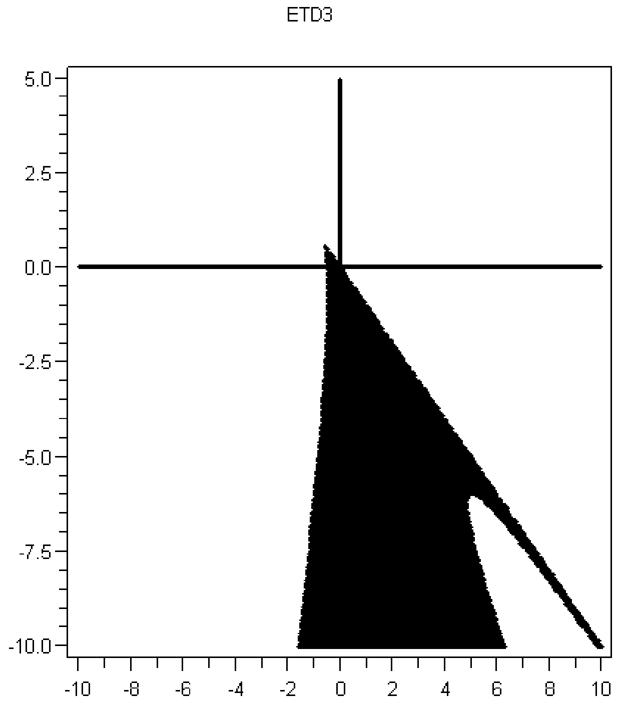


FIGURE 3.4: Stability regions (shaded) in the real (x, y) plane for the ETD1 (3.14) and the ETD2 (3.15) methods.

the curves where the boundary passes through the point $x = -y$. Finally for values of $y = -3, \ldots, -\infty$ the stability regions are the interior of those portions of the curves that contain the origin, see figure 3.1 for $y = -10, -15$.

In the real (x, y) plane, the right-hand boundary for both the ETD1 (3.14) and the ETD2 (3.15) schemes, corresponding to substituting $r = 1$ in equations (3.25) and (3.26) respectively, is the line $x+y = 0$. The left-hand boundaries for the ETD1 and the ETD2 schemes, corresponding to substituting $r = -1$ in equations (3.25) and (3.26) respectively, are the curve

$$x = \frac{-y(e^y + 1)}{e^y - 1},$$



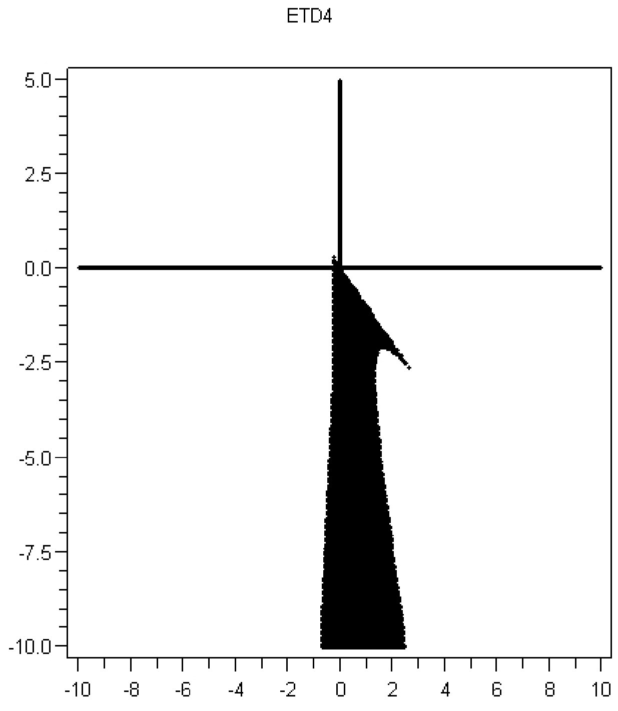


Figure 3.5: Stability regions (shaded) in the real (x, y) plane for the ETD3 (3.16) and the ETD4 (3.17) methods.

and the curve [19]

$$x = \frac{-y^2(e^y+1)}{(y+2)e^y - 3y - 2},$$

respectively.

Similarly, the right-hand boundary for both the third and fourth-order ETD methods, corresponding to $r = 1$, is the line $x + y = 0$, but only for the specified values of y in the ranges mentioned previously (for values outside the ranges, there is no simple formula for the right-hand boundaries due to the complicated structure of the stability regions, and so, the plots of the stability regions of the ETD3 and the ETD4 methods in the real (x, y) plane are produced using Maple). For $r = -1$, the left-hand boundaries for the ETD3 (3.16) and the ETD4 (3.17) methods are the curves

$$x = \frac{-y^3(e^y+1)}{(y^2+4y+4)e^y - 7y^2 - 8y - 4},$$

and

$$x = \frac{-3y^4(e^y+1)}{(3y^3+20y^2+36y+24)e^y - 45y^3 - 68y^2 - 60y - 24},$$

respectively.

The stability regions for the ETD1 and the ETD2 schemes are illustrated in figure 3.4, whereas those for the ETD3 and the ETD4 methods are illustrated in figure 3.5. Note that the horizontal and the vertical axes in these figures represent Re(x) and Re(y) respectively. In line with the previous case of the complex x plane, we find that the stability region of the ETD1 method is broader than those of the other three higher-order methods, whereas it is very narrow for the ETD4 method. Additionally, we find that the region of stability for all schemes includes the negative y-axis [19] and grows larger as y decreases.

3.3.2 Stability of RK Exponential Time Differencing Methods

The basic question of stability is again addressed by applying each of the ETD-RK schemes to the linearized problem (3.24), and determining the boundary separating growing and decaying solutions u_n.

When applying the ETD2RK1 method (3.19) to (3.24), we obtain

$$\begin{aligned} u_{n+1} = {} & e^{c\Delta t}u_n + \{((c\Delta t - 1)e^{c\Delta t} + 1)\lambda u_n \\ & + (e^{c\Delta t} - c\Delta t - 1)(\lambda e^{c\Delta t} + \lambda^2(e^{c\Delta t} - 1)/c)u_n\}/(c^2\Delta t). \end{aligned}$$

Defining $r = u_{n+1}/u_n$, $x = \lambda\Delta t$ and $y = c\Delta t$, leads to [19]

$$r = e^y + \left(\frac{e^y - 1}{y}\right)^2 x + \left(\frac{(e^y - 1)(e^y - y - 1)}{y^3}\right) x^2. \tag{3.27}$$

Similarly, for the ETD2RK2 scheme (3.20), r satisfies

$$r = e^y + \left(\frac{2(e^y - y - 1)e^{y/2} + (y-2)e^y + y + 2}{y^2}\right) x + \left(\frac{2(e^y - y - 1)(e^{y/2} - 1)}{y^3}\right) x^2. \tag{3.28}$$

In a similar way, when applying the third-order ETD3RK (3.21) and the fourth-order ETD4RK (3.22) schemes to (3.24), the equation is linear for the factor r for each scheme. The formulas for the factor r for the ETD3RK and the ETD4RK methods are not neat and simple, hence, they are not given explicitly here.

We note that the equations for the factor r are different for the different formulas of an s-order ETD-RK scheme. This is in contrast to the fact that the different formulas of an explicit s-order ($s = 1, 2, 3, 4$) RK scheme have the same equation for the factor r [84], that is $r = |1+y+y^2/2+\cdots+y^s/s!|$. This curve for r is obtained by applying an explicit s-order RK scheme to the linearized problem $du(t)/dt = cu(t)$ where $r = u_{n+1}/u_n$ and $y = c\Delta t$.

In the limit $y \to -\infty$, equations (3.27) and (3.28) become

$$\frac{x^2}{y^2} \approx r, \tag{3.29}$$

for the ETD2RK1 scheme, and

$$\frac{2x^2}{y^2} + \frac{x}{y} \approx r, \tag{3.30}$$

for the ETD2RK2 scheme, respectively, and the equations for the factor r for the ETD3RK (3.21) and the ETD4RK (3.22) schemes become

$$-\frac{2x^3}{y^3} - \frac{x^2}{y^2} \approx r, \tag{3.31}$$

and

$$\frac{2x^4}{y^4} - \frac{x^2}{y^2} \approx r, \tag{3.32}$$

respectively.

Our analysis below depends on choosing real negative values of the constant c and looking for a region of stability in the complex x plane where the solution u_n remains bounded as $n \to \infty$. The boundary of the stability region is determined by writing $r = e^{i\theta}$, $\theta \in [0, 2\pi]$ in each equation for the factor $r = u_{n+1}/u_n$ in the

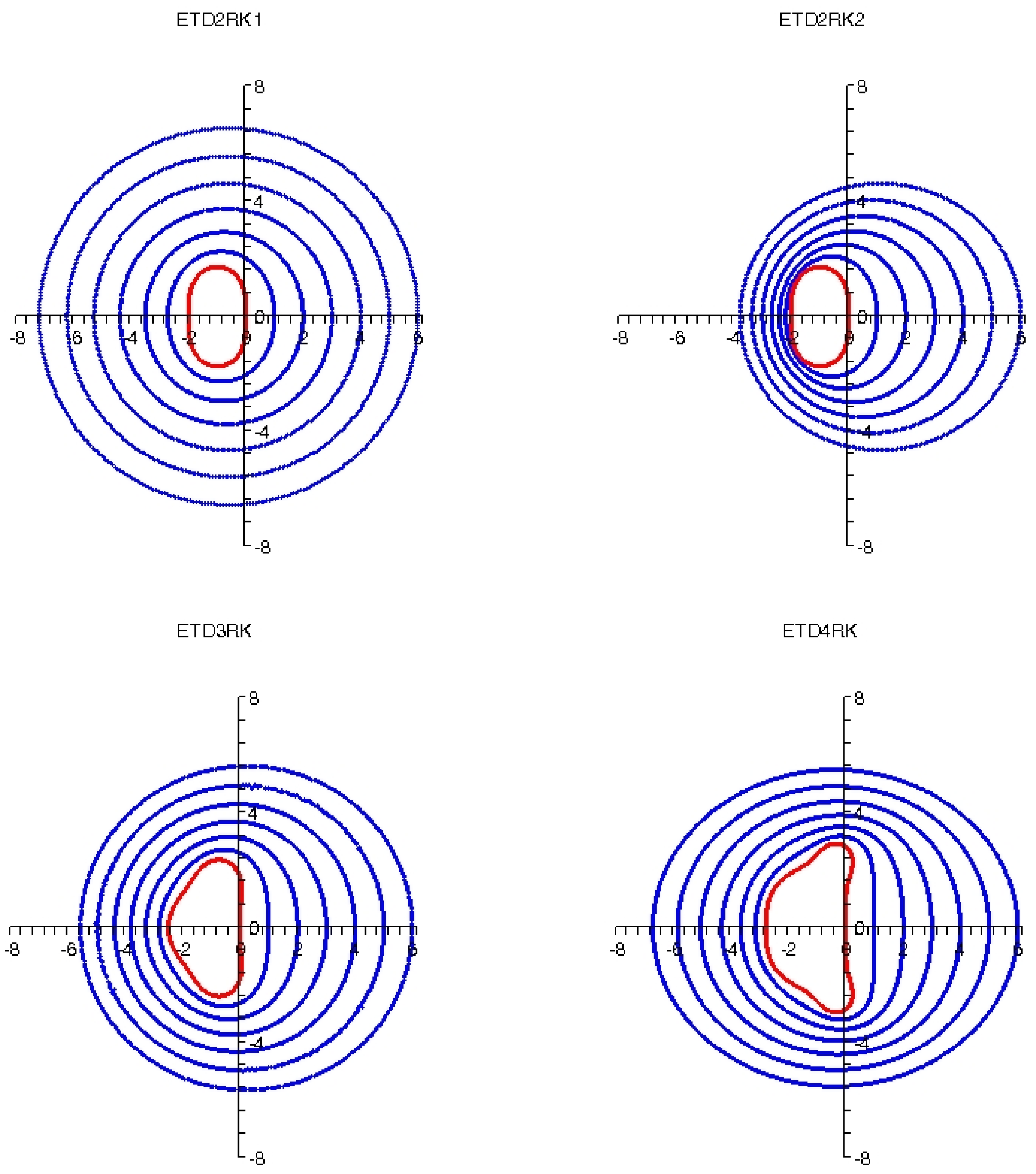


FIGURE 3.6: Stability regions in the complex x plane. The curves correspond to $y = -6, -5, -4, -3, -2, -1$, from the outer curve to the inner curve respectively. The inner red curves correspond to $y = 0$.

ETD2RK1 (3.19), the ETD2RK2 (3.20), the ETD3RK (3.21) and the ETD4RK (3.22) methods and then by solving for $x = \lambda\Delta t$. The corresponding families of stability regions are plotted in the complex x plane and displayed in figures 3.6, 3.7 and 3.8. Note that, in these figures, the horizontal and the vertical axes represent

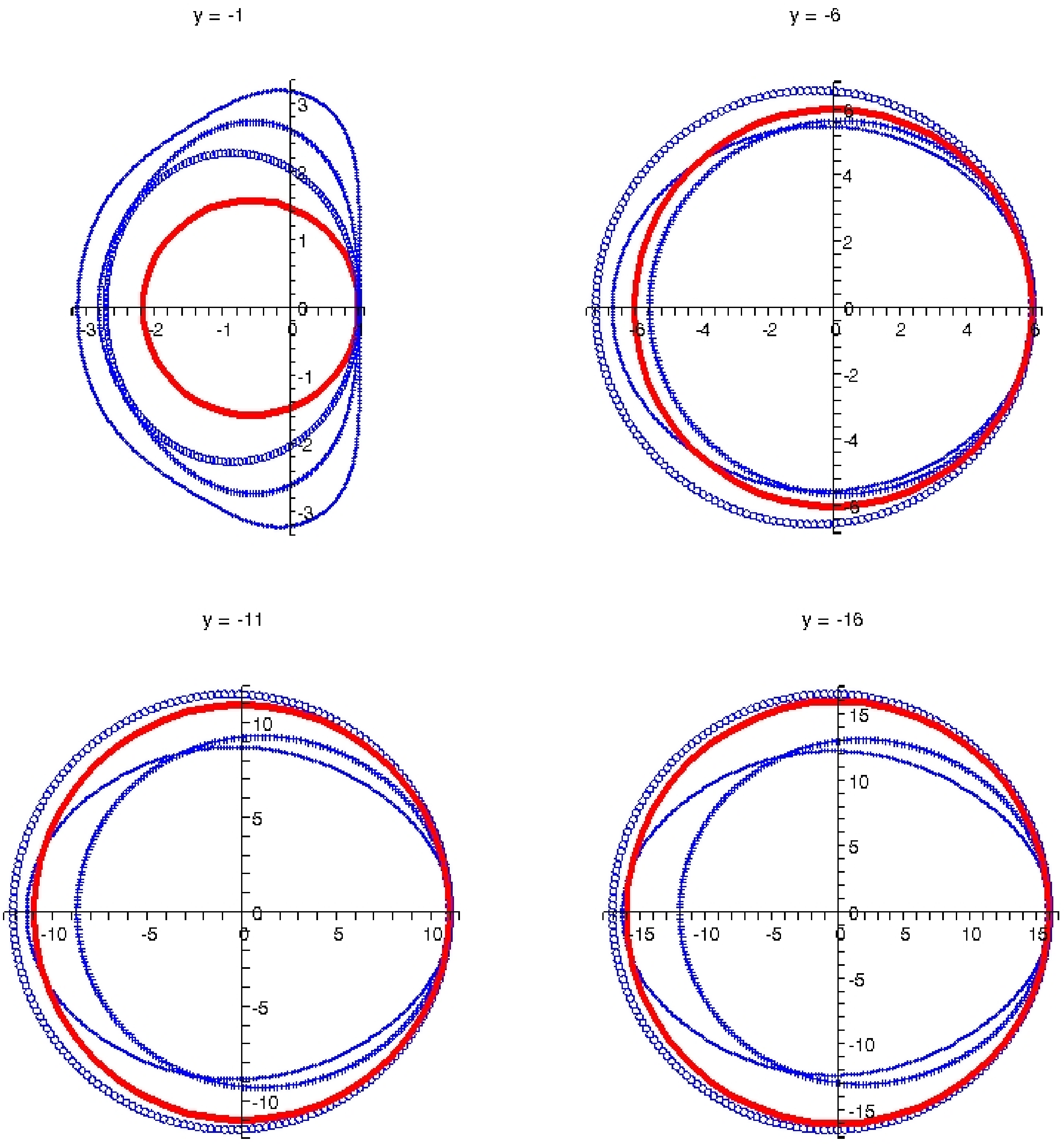


FIGURE 3.7: Stability regions in the complex x plane for different values of y. The four methods are: ETD1 (red-solid), ETD2RK1 (circle), ETD3RK (cross), ETD4RK (point).

Re(x) and Im(x) respectively.

Figure 3.6 exhibits the complex plot of the stability regions of the four schemes with different values of y. The outer curves correspond to $y = -6$ and the inner blue curves correspond to $y = -1$. Generally, the stability regions of the ETD-RK

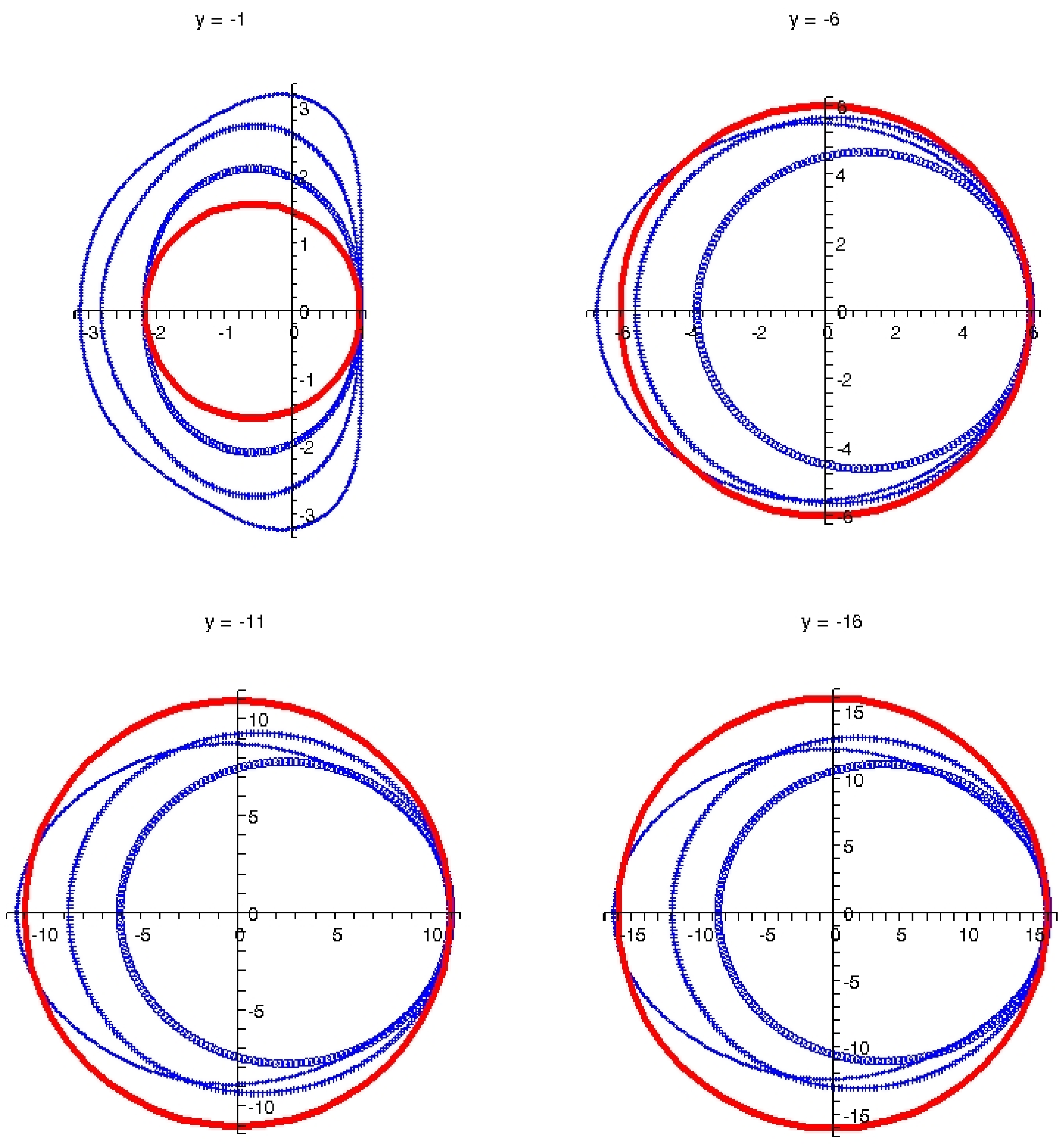


FIGURE 3.8: Stability regions in the complex x plane for different values of y. The four methods are: ETD1 (red-solid), ETD2RK2 (circle), ETD3RK (cross), ETD4RK (point).

schemes are larger than those of the explicit multi-step ETD schemes. We also find that for all schemes, the origin belongs to the stability regions and the boundary of the stability regions passes through the point $x = -y$ [19, 23]. This is consistent with the true stability boundary of the differential equation (3.24) of the test problem, namely, the solution decays for $Re(c+\lambda) < 0$ and it grows otherwise. Clearly, as shown in figure 3.6, the region of stability for all ETD-RK schemes grows larger as y decreases. The red curves correspond to the case $y = 0$, where the stability regions of the ETD-RK schemes coincide with the those of the corresponding order RK schemes [84]. This is expected since in the limit $y \to 0$, ETD-RK schemes reduce to the corresponding order explicit RK schemes (this result was also found in [23] for up to fourth-order ETD-RK schemes and by **Krogstad** [49] for the ETD4RK and the ETDRK4-B methods). Note that the stability regions of the RK schemes increase as the order of the methods increases. Also note the fact that the different formulas of an explicit s-order RK scheme have the same stability regions [84]. However, we find, as shown in figure 3.6, that the stability region of the ETD2RK2 scheme is smaller than that of the ETD2RK1 scheme for any given value of y, and thus, generally the different formulas of an s-order ETD-RK scheme do not have the same stability region.

Figure 3.7 illustrates the plot of the ETD1, the ETD2RK1, the ETD3RK and the ETD4RK methods all in one diagram for different values of y. For $y = -1$, we find that the stability region increases as the order of the ETD-RK schemes increases, that is, the ETD2RK1 scheme (3.19) has the smallest stability region while the ETD4RK scheme (3.22) has the largest. As y decreases, see figure 3.7, we find that the stability regions of the ETD1 and the ETD2RK1 schemes become slightly larger than those of the other schemes. As $y \to -\infty$, the explicit ETD1 and ETD2RK1 schemes coincide in their stability regions and become the largest, and simultaneously simplify to the disc $|x| < |y|$ [19] (this corresponds to substituting $|r| = 1$ in equation (3.29)), while the stability region of the ETD3RK scheme becomes the smallest.

Figure 3.8 illustrates the plot of the ETD1, the ETD2RK2, the ETD3RK and the ETD4RK methods all in one diagram for different values of y. We notice that as the order of these schemes increases the stability regions increase in size for any given value of y. Furthermore, as $y \to -\infty$ the stability region of the ETD4RK scheme contains those of the ETD3RK and the ETD2RK2 schemes of which the latter is the smallest, though, the stability region of the ETD4RK schemes is contained by those of the ETD2RK1 and the ETD1 schemes [23], see figures 3.7 and 3.8.

In the real (x, y) plane, the left-hand boundaries for the ETD2RK1 (3.19) and ETD2RK2 (3.20) schemes, corresponding to substituting $r = 1$ in equations (3.27)

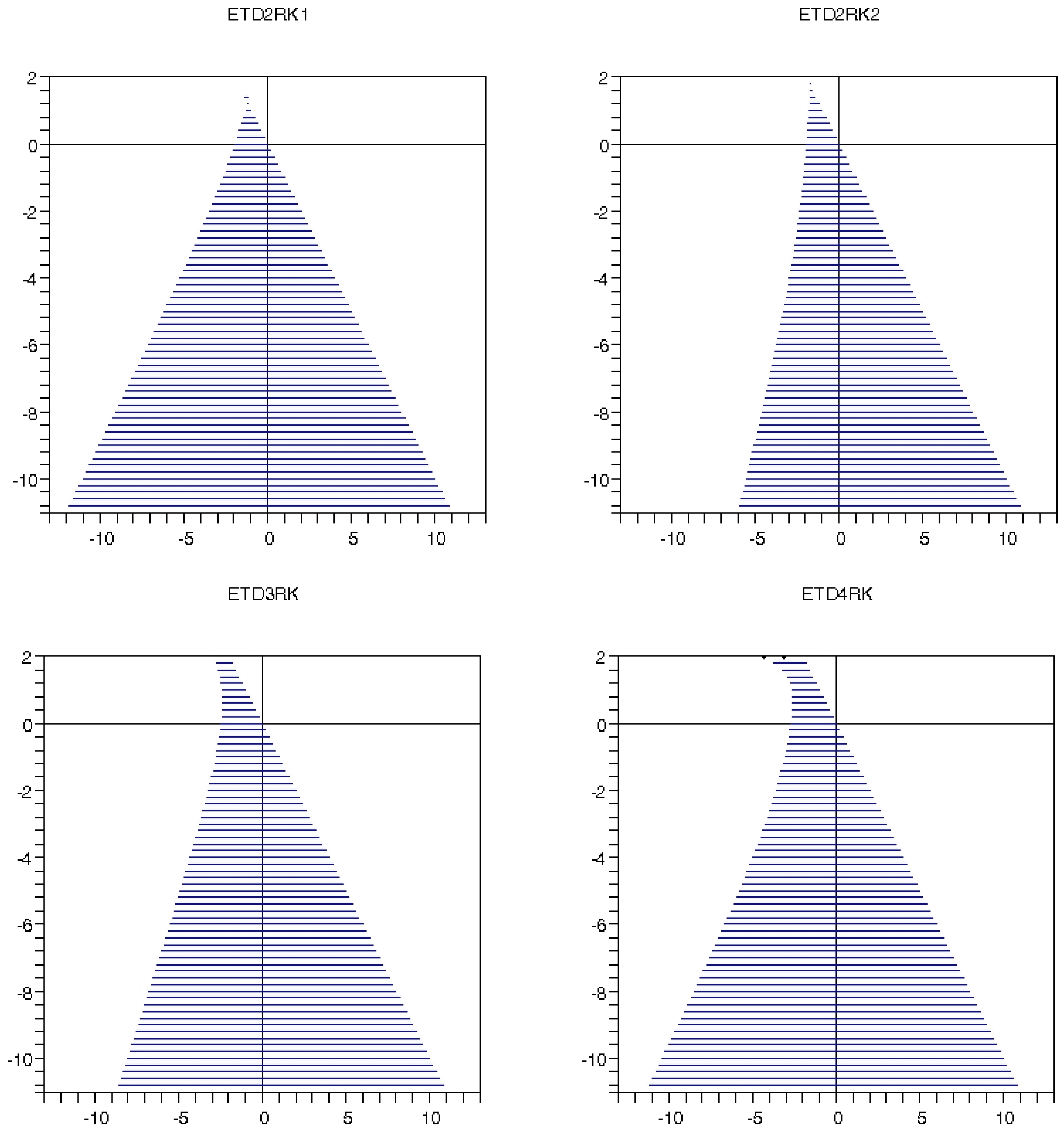


Figure 3.9: Stability regions (shaded) in the real (x, y) plane for four methods.

and (3.28) respectively[1], are the curve [19]

$$x = \frac{-y^2}{e^y - y - 1},$$

[1]No stability boundaries corresponding to $r = -1$ are obtained for the ETD2RK1, ETD2RK2 and ETD4RK schemes.

and the curve

$$x = \frac{-y^2(e^{y/2}+1)}{2(e^y - y - 1)},$$

respectively, while in the same situation, the right-hand boundary for both schemes is the line $x + y = 0$.

Similarly, the equation for the factor r is a third and fourth order polynomial in x for the ETD3RK (3.21) and the ETD4RK (3.22) schemes respectively, and for both schemes, the right-hand boundaries (corresponding to $r = 1$) are the line $x + y = 0$ (note that the right-hand boundary is the same for all four schemes [23]). The formulas of the left-hand boundary for the ETD3RK and the ETD4RK methods, corresponding to $r = -1$ and $r = 1$ respectively, are complicated and hence not given explicitly. The real stability regions for these four methods are shown in figure 3.9, where the horizontal and the vertical axes represent $\mathrm{Re}(x)$ and $\mathrm{Re}(y)$ respectively.

In figure 3.9, we notice that the region of stability for all schemes includes the negative y axis [19] and grows larger as y decreases. In addition, the stability region of the ETD2RK1 scheme is broader than that of the ETD2RK2 scheme, which agrees with the previous case of the complex x plane.

In the limit $y \to -\infty$, the right-hand boundaries for the ETD2RK1 (3.19) [19], the ETD2RK2 (3.20) and the ETD4RK (3.22) schemes correspond to

$$x \approx -y,$$

which is the same for all schemes. The left-hand boundaries for the ETD2RK1, the ETD2RK2 and the ETD4RK schemes, corresponding to substituting $r = 1$ in equations (3.29), (3.30) and (3.32) respectively, are

$$x \approx y,$$

and

$$\left(\frac{2x}{y} - 1\right)\left(\frac{x}{y} + 1\right) \approx 0 \Longrightarrow x \approx \frac{y}{2},$$

and

$$(2x^2 + y^2)(x^2 - y^2) \approx 0 \Longrightarrow x \approx y,$$

respectively, see figure 3.9. In the same limit, the right-hand and the left-hand boundaries for the ETD3RK (3.21) scheme, corresponding to substituting $r = 1$ and $r = -1$ respectively in equation (3.31), are

$$(2x^2 - xy + y^2)(x + y) \approx 0 \Longrightarrow x \approx -y,$$

and

$$x \approx 0.657y,$$

respectively.

3.4 Conclusion

In our study of the ETD methods we have found that these methods possess the following features:

- If the nonlinear part $F(u(t), t)$ of the differential equation (3.3) is zero, the integrator produces the exact solution to the ODE and so is automatically A-stable.
- If the linear part is zero ($c = 0$ in (3.3)), the ETD and the ETD-RK integrators reduce to linear multi-step or classical explicit Runge-Kutta methods respectively.

We have also discussed the stability properties of the ETD and ETD-RK schemes up to fourth-order. We have found that the various formulas of the explicit second-order ETD-RK schemes do not have the same stability region, in contrast to the fact that all RK schemes of a given order have the same stability region. In addition, we have found that as the order of the ETD-RK methods increases the stability regions increase in size, i.e. the stability region of the ETD4RK scheme contains those of the ETD3RK and the ETD2RK2 methods. However, as the stiffness parameter $c \to -\infty$ in (3.23), the stability region of the ETD2RK1 method contains those of the ETD4RK, the ETD3RK and the ETD2RK2 methods. This is in contrast to noted properties of the stability regions of the multi-step ETD methods. We have found that the stability region of the fourth-order ETD scheme is contained by those of the lower order ones, i.e. the stability regions of the ETD methods shrink as the order of the methods increases. In general, we have found that the stability regions of the ETD-RK methods are larger than those of the multi-step ETD methods.

To conclude, the stability characteristics of the ETD and the ETD-RK methods (the stability regions grow larger as the stiffness parameter $c \to -\infty$) reveal that when solving stiff problems the selection of the time step size for these methods is only limited by accuracy and not stability. This indicates the possibility of using a large time step and consequently these methods provide computational savings over conventional explicit methods.

Chapter 4

Various Algorithms for Evaluating the ETD Coefficients

Outline of Chapter

The coefficients of the Exponential Time Differencing (ETD) methods are the exponential and related functions of the linear operators of a semi-discretized partial differential equation (PDE). When applying the ETD methods, the computations of the coefficients need only to be carried out once at the start of the integration if a constant time step is used during the integration. The computation of these functions depends significantly on the structure and the range of the eigenvalues of the linear operator and the dimensionality of the semi-discretized PDE. The linear part should not be explicitly time dependent and, if possible, should be represented as a diagonal matrix in order for the exponential integrators to be computationally competitive. On the other hand, the linear part might have eigenvalues equal to or approaching zero, which leads to complications in the computation of the coefficients.

In this chapter, we discuss methods for the accurate computation of the ETD coefficients and the efficiency of implementation. We first explain why the ETD methods need further development, and then address ourselves to describing the various algorithms. We analyze their performance and their computational cost, and weigh their advantages for improving the numerical difficulties in approximating the ETD coefficients. This gives us the chance to distinguish between the algorithms and choose the one that is best for the success of the methods.

4.1 Introduction

When a stiff partial differential equation (PDE) with periodic boundary conditions is discretized in space using Fourier spectral methods [25, 83, 84] (see §2.3), a system of coupled ordinary differential equations (ODEs) in time t, for the Fourier coefficients, is obtained. The linear part of this system is represented by a diagonal matrix in the Fourier basis, which might have eigenvalues of both large and small magnitude. A complication [19] arises in using the time discretization ETD methods (see §3) for problems which have eigenvalues equal or close to zero in the diagonal linear operator. These difficulties are twofold: firstly, when some of the eigenvalues are equal to zero, the explicit formulas (3.12) for the coefficients g_m cannot be used directly since they involve division by zero, i.e. $c = 0$. Instead, the limiting form of the coefficients as $c \to 0$ must be used. Secondly, these methods suffer from rounding errors occurring due to the large amount of cancellation in the ETD coefficients g_m (3.12) for eigenvalues approaching zero.

To identify the problem, consider evaluating numerically the expression

$$f_1(z) = \frac{e^z - 1}{z}, \tag{4.1}$$

that appears in the ETD1 scheme (3.14), for z a scalar. $f_1(z)$ is an analytic function and has a removable singularity at $z = 0$. The limiting form of the expression $f_1(z)$ as $z \to 0^{\pm}$ should result in 1. Undesirably, as z gets close to zero, the expression does not approach 1 when evaluated numerically. The terms in the expression do not cancel precisely and the small errors of cancellation become substantial as we are dividing the result by a number approaching zero. This problem gets worse in higher order methods.

The expressions

$$f_k(z) = \frac{e^z - G_k(z)}{z^k}, \qquad k = 1, 2, \ldots, s, \tag{4.2}$$

where

$$G_k(z) = \sum_{j=0}^{k-1} \frac{z^j}{j!}, \tag{4.3}$$

is the first k terms in the Taylor series approximation to the exponential function $f_0(z) = e^z$, are at the core of the ETD and ETD-RK methods (3.13) of order s. In fact the coefficients of these methods are really a combination of the expressions $f_k(z)$ (4.2). As with $f_1(z)$ (4.1), the expressions $f_k(z)$ for $k > 1$ suffer from numerical evaluation errors as $z \to 0^{\pm}$. In fact, in the limiting form of the expressions as $z \to 0^{\pm}$, the numerator and denominator are of $O(z^k)$. Hence, in order to implement the

ETD and ETD-RK methods accurately, we need an accurate algorithm to evaluate the f_k.

Figure 4.1 shows a plot of the exponential function $f_0(z)$, the formulas $f_1(z)$ (4.1), and

$$f_2(z) = \frac{e^z - 1 - z}{z^2}, \tag{4.4}$$

and

$$f_3(z) = \frac{(e^z - 1 - z - z^2/2)}{z^3}, \tag{4.5}$$

for values of z over the range $[-2, 2]$. Analytically, for values of $z \to \infty$, $f_1(z) \approx e^z/z$, $f_2(z) \approx e^z/z^2$ and $f_3(z) \approx e^z/z^3$ and generally, as $z \to \infty$

$$f_k(z) \approx \frac{e^z}{z^k}, \quad k = 1, 2, 3, \ldots, s. \tag{4.6}$$

Also, for values of $z \to 0^\pm$, $f_1(z) \approx 1$, $f_2(z) \approx 1/2$ and $f_3(z) \approx 1/6$ and in general, as $z \to 0^\pm$

$$f_k(z) \approx \frac{1}{k!}, \quad k = 1, 2, 3, \ldots, s. \tag{4.7}$$

And as $z \to -\infty$, $f_1(z) \approx -1/z$, $f_2(z) \approx -1/z$ and $f_3(z) \approx -1/(2z)$ and in general, as $z \to -\infty$

$$f_k(z) \approx \frac{-1}{(k-1)!z}, \quad k = 1, 2, 3, \ldots, s. \tag{4.8}$$

Numerically however, as z gets close to zero, these formulas $f_k(z)$ (4.2) suffer from serious cancellation errors[1], as we are dividing the result by a number approaching zero and raised to a power, see for example the plot of the function $f_3(z)$ (4.5) in figure 4.2 over a range of very small values in magnitude of z.

In order to make the ETD and ETD-RK methods (3.13) practical in this case (where the linear part of a discretized PDE is represented by a diagonal matrix in the Fourier basis) only the scalar form of $f_k(z)$ (4.2) is required, since the exponential of a diagonal matrix can be obtained by just exponentiating every entry on the main diagonal independently. A simple approach here is to use a Taylor series expansion [19] to approximate such expressions for values of z less than some chosen threshold value z_{th}, as follows

$$f_k(z) = \sum_{j=k}^{\infty} \frac{z^{j-k}}{j!}, \quad z \leq z_{th}, \; k = 1, 2, \ldots, s, \tag{4.9}$$

[1]These errors depend on the computer precision. So, in single computer precision these errors get worse.

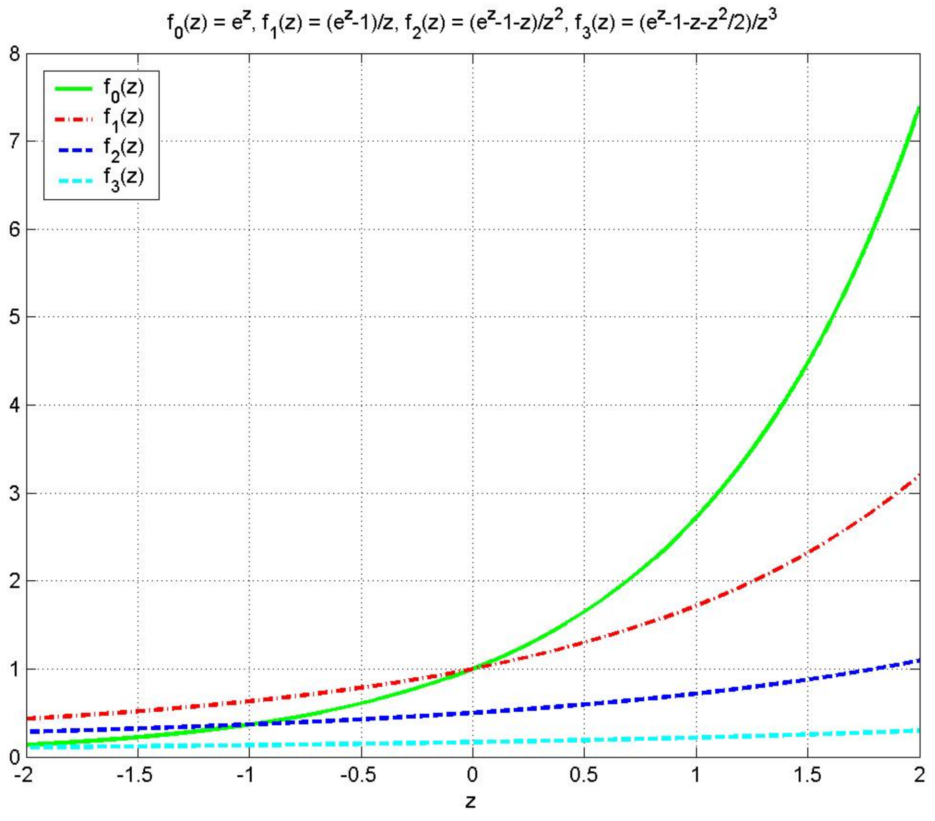


FIGURE 4.1: The values of the exponential function $f_0(z)$ and the function $f_k(z)$ (4.2) of orders $k = 1, 2, 3$ versus the values of z.

and to use the explicit formulas of the ETD coefficients g_m (3.12) for values of z larger than z_{th}.

On the other hand, if we discretize a PDE in space using finite difference formulas [58, 83] (see §2.2) or Chebyshev polynomials [11, 25, 83, 84], for instance, a system of coupled ODEs is obtained. Thus, the linear operator is represented by a non-diagonal matrix that might have eigenvalues with values of both large and small magnitude for stiff problems. Applying the ETD methods here requires the computation of a non-diagonal matrix exponential, which in itself is not a straightforward task [60]. Furthermore, having eigenvalues equal to or close to zero in the non-diagonal matrix, again leads to inaccuracies in evaluating expressions of the form (4.2). In this case we cannot distinguish between eigenvalues of small and large magnitude and simply switch between using the Taylor series expansion and the explicit formulas of the ETD coefficients (3.12) respectively. It is therefore important to have an accurate numerical algorithm for evaluating the function $f_k(z)$ (4.2) in both scalar and non-diagonal matrix cases. One would like a single algo-

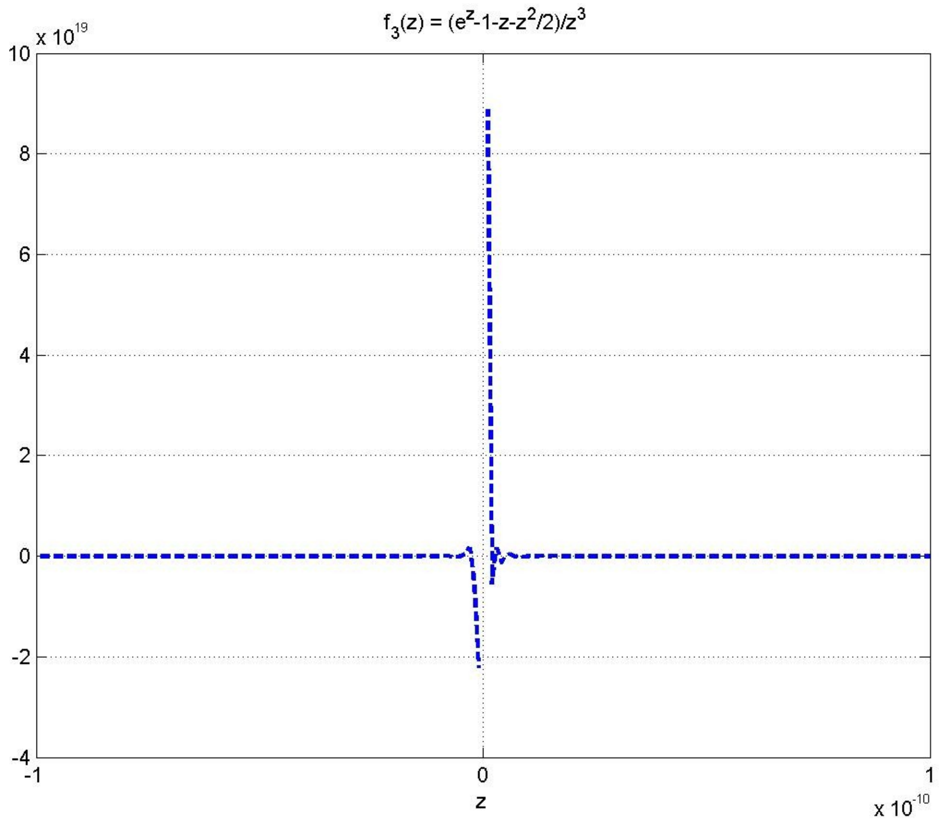


FIGURE 4.2: The values of the function $f_3(z)$ (4.5) versus a range of very small values in magnitude of z, evaluated numerically with 16-digit precision.

rithm that is simultaneously accurate for all values of z in the scalar case, and also performs well in the non-diagonal matrix cases.

In §4.2, we describe some of the algorithms that appear to be practical for approximating the expression $f_k(z)$ (4.2) of orders $k = 1, 2, 3$, since these expressions are the most frequently used in the ETD methods (3.13), for the scalar z with values of large and small magnitude. Then in §4.3 we set up some tests on the second-order centered difference differentiation matrix (see §2.2) for the second derivative, to represent the non-diagonal matrix case. We also conduct similar tests on the Chebyshev differentiation matrix for the second derivative and the second-order centered difference differentiation matrix for the first derivative, in §4.4 and §4.5 respectively. The aim is to show that the algorithms also work well for these non-diagonal matrices, and that their efficiency is by no means restricted to any special structure of certain matrices.

The algorithms considered are Taylor series, an algorithm based on the Cauchy Integral Formula [44, 45], different forms of the Scaling and Squaring algorithms

[8, 9, 35, 37, 47, 53, 60, 76], the Composite Matrix algorithm [2, 54, 67], and the Matrix Decomposition algorithm [60] for non-diagonal matrix cases. We assess the effectiveness of these algorithms by considering their stability, accuracy, efficiency, ease of use and simplicity. The accuracy of an algorithm refers primarily to the error introduced by the algorithm. Efficiency of the algorithms is measured by the amount of computer time required to approximate such expressions, and this is the primary focus of §4.6. Also, we outline the issues that contribute to our understanding of the limitations of the algorithms if they fail to produce accurate enough results. The overall conclusions of the comparison tests are given in §4.7.

4.2 The Scalar Case

To demonstrate the effectiveness of the algorithms, we test them against each other for the scalar z with values of large and small magnitude to approximate the expression $f_k(z)$ (4.2) of orders $k = 1, 2, 3$. We compute the relative error of each algorithm, given by

$$\textit{relative error} = \frac{|\textit{exact value} - \textit{approximate value}|}{|\textit{exact value}|}, \tag{4.10}$$

where the exact values of the expressions were approximated using 50 digit arithmetic in Matlab code. Figures 4.3 and 4.4 show the relative errors of each algorithm to approximate the expressions $f_1(z)$ (4.1) and $f_3(z)$ (4.5) versus the values of z (we find that for $f_2(z)$ (4.4), the algorithms behave in a qualitatively similar way to $f_3(z)$). The figures also show the errors for the use of the explicit formulas; this means simply evaluating the formulas $f_1(z)$ and $f_3(z)$ with standard double precision (16 digits) arithmetic.

4.2.1 Taylor Series

The formula

$$e^z \approx 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \cdots + \frac{z^m}{m!},$$

for some integer m may be used to approximate the exponential in the expression $f_k(z)$ (4.2) of orders $k = 1, 2, 3$, so that $f_k(z)$ becomes

$$f_k(z) \approx \sum_{j=k}^{m-1} \frac{z^{j-k}}{j!}.$$

Attention to where to truncate the series is important if efficiency is being considered. We can simply sum the series until adding another term does not alter the accuracy of the algorithm.

In doing the test described in §4.2, we find that for $|z| \ll 1$, the explicit formulas for $f_1(z)$ (4.1) and $f_3(z)$ (4.5) are imperfect due to the cancellation errors, but the Taylor expansion with 30 terms is remarkably good, see figures 4.3 and 4.4. For $|z| \gg 1$, the explicit formulas $f_1(z)$ and $f_3(z)$ give acceptable results but the Taylor expansion is imprecise. For $z \ll -1$, the errors in using the Taylor series are due to the "catastrophic cancellation". This term refers to the extreme loss of accuracy when small numbers are additively computed from large numbers. The errors in using the Taylor expansion can actually be larger than the correct exponential, and the answer will not be correct, no matter how many terms in the series are summed (it should be emphasized here, that the difficulty is not the truncation of the series, but the truncation of the arithmetic). In the limit $z \to \infty$, see figure 4.3, the numerical relative errors for the Taylor approximation approach 1 since the exponential values in the explicit formulas $f_1(z)$ and $f_3(z)$ dominate over the Taylor expansion.

The primary advantage of this algorithm is the simplicity and ease of implementation. Note that figures 4.3 and 4.4 show that, in the scalar case, it is possible to use the Taylor series for $|z| < 1$ and the explicit formulas for $f_1(z)$ (4.1) and $f_3(z)$ (4.5) for $|z| \geq 1$, without significant loss of accuracy.

4.2.2 The Cauchy Integral Formula

To overcome the numerical difficulties in the ETD and ETD-RK methods (3.13) of order s, a different tactic for evaluating the function $f_k(z)$ (4.2) of orders $k = 1, 2, \dots, s$ was proposed by **Kassam** and **Trefethen** in [44, 45]. The key idea is to approximate the functions (for matrices or scalars) by means of contour integrals in the complex plane.

The well-known **Cauchy Integral Formula** [55]

$$f(z) = \frac{1}{2\pi i} \int_\Gamma \frac{f(T)}{T - z} dT, \tag{4.11}$$

evaluates the analytic function f via an integral along a closed contour Γ that encloses z. The Cauchy integral formula (4.11) says that the values of f on Γ completely determine the values of f inside Γ.

The simplest choice of the contour Γ is a circle with radius R centered at some point z_0,

$$\Gamma = \{T(\theta) = z_0 + Re^{i\theta} :\ 0 \leq \theta \leq 2\pi\}.$$

Then by the definition of the contour integral for any function H

$$\int_\Gamma H(T)dT = \int_\theta H(T(\theta))dT(\theta)d\theta, \tag{4.12}$$

the Cauchy integral (4.11) of a function of a scalar z along the circular contour Γ becomes

$$f(z) = \frac{1}{2\pi i}\int_0^{2\pi} \frac{f(z_0 + Re^{i\theta})}{T(\theta) - z} Rie^{i\theta} d\theta = \frac{1}{2\pi}\int_0^{2\pi} \frac{f(z_0 + Re^{i\theta})}{T(\theta) - z}(T(\theta) - z_0)d\theta, \tag{4.13}$$

which is a periodic integral of our function evaluated at points on the circumference of the circular contour. If we employ the periodic **Trapezium Rule** defined by

$$\int_0^{2\pi} P(\theta)d\theta \approx \frac{2\pi}{N}\sum_{j=1}^{N} P(\theta_j),\ \theta_j = \frac{2\pi j}{N}, \tag{4.14}$$

to approximate the integral on the right-hand side of (4.13), we obtain the formula proposed by **Kassam** and **Trefethen** [44, 45] for a circular contour,

$$f(z) \approx \frac{1}{N}\sum_{j=1}^{N} \frac{f(T(\theta_j))}{T(\theta_j) - z}(T(\theta_j) - z_0). \tag{4.15}$$

Referring to [44, 45, 84], the authors stated that the periodic Trapezium Rule is simply the Fourier spectral method for integrating a periodic function. The convergence of spectral methods in general, and Fourier methods in particular, depends on the smoothness of the function that is being interpolated. For analytic functions, the Fourier coefficients decay exponentially [79] and we have correspondingly exponential convergence of spectral methods, including the periodic Trapezium Rule.

This algorithm has turned out to be very powerful, as figures 4.3 and 4.4 show. Testing the algorithm as described in §4.2 shows that the algorithm performs very well when approximating the expression $f_k(z)$ (4.2) of orders $k = 1, 3$ for the scalar z with values of large and small magnitude (qualitatively similar results are found for $k = 2$). For each value of z, the chosen contour is a circle of radius $R = 1$, centered at $z_0 = z$, and sampled at $N = 32$ equally spaced points $\{\theta_j\}$, and $f_k(z)$ is approximated by (4.15) as follows

$$f_k(z) \approx \frac{1}{N}\sum_{j=1}^{N} f_k(T(\theta_j)), \tag{4.16}$$

which is an average of the function values at the N points $T(\theta_j) = z + Re^{i\theta_j}$ around the discretized circumference of the circle (it is important to ensure that none of the points on the contour are close to or at the origin, otherwise the original problem of rounding errors reappears).

In the case where the linear part of a discretized PDE is represented by a diagonal matrix in the Fourier basis which may have eigenvalues that are zero or of small magnitude, we can compute the expression $f_k(z)$ (4.2) of orders $k = 1, 2, \ldots, s$ for each element on the diagonal independently by again using the above formula (4.16) [44, 45], for circles centered at each element on the matrix diagonal.

4.2.3 Scaling and Squaring Algorithm: Type I

One of the most widely used of the algorithms that have been proposed for approximating expressions such as $f_k(z)$ (4.2), $k = 1, 2, \ldots, s$ that appear in the ETD and ETD-RK methods (3.13) of order s, is the Scaling and Squaring algorithm [8, 9, 35, 37, 47, 53, 60, 76]. This section considers the algorithm in the form in which we scale up from small values of $|z|$; the alternative approach of scaling down from large values of $|z|$ is discussed in §4.2.4 (for ease of presentation we outline the theory for the scalar case, but the algorithm is equally applicable to matrices, to be described in §4.3.4).

Consider first the accurate evaluation of the exponential function, $f_0(z) = e^z$. It is possible to use the Taylor series or Padé approximation (4.94) [35, 37, 47, 76] for $|z| \leq 1$, but to avoid loss of accuracy for $|z| > 1$, we use the Scaling and Squaring algorithm [60], based on the identity

$$f_0(2z) = (f_0(z))^2 = (e^z)^2. \tag{4.17}$$

First we compute $f_0(2^{-l}z)$ for some l chosen to be the smallest integer such that

$$l \geq \frac{\log(|z|/\delta)}{\log 2}, \tag{4.18}$$

so that for some threshold value δ we have $|2^{-l}z| \leq \delta$. This computation is efficiently and accurately performed using the Taylor expansion or Padé approximation. Using (4.17), the resulting value is then squared l times to obtain the final answer

$$f_0(z) = [f_0(2^{-l}z)]^{2^l}. \tag{4.19}$$

A similar approach can be used for computing the expression $f_k(z)$ (4.2) of orders $k = 1, 2, 3$. The algorithm uses either the identities (taken from [9])

$$f_1(2z) = \frac{1}{2}\left(f_0(z)f_1(z) + f_1(z)\right), \tag{4.20}$$

$$f_2(2z) = \frac{1}{4}\left(f_1(z)f_1(z) + 2f_2(z)\right), \tag{4.21}$$

$$f_3(2z) = \frac{1}{8}\left(f_1(z)f_2(z) + f_2(z) + 2f_3(z)\right), \tag{4.22}$$

or the identities (taken from [54])

$$f_1(2z) = \frac{1}{2}\left(f_0(z)f_1(z) + f_1(z)\right), \tag{4.23}$$

$$f_2(2z) = \frac{1}{4}\left(f_0(z)f_2(z) + f_1(z) + f_2(z)\right), \tag{4.24}$$

$$f_3(2z) = \frac{1}{8}\left(f_0(z)f_3(z) + \frac{1}{2}f_1(z) + f_2(z) + f_3(z)\right). \tag{4.25}$$

A general form of the squaring relations (4.23) - (4.25) stated with proof in [76] is

$$f_k(2z) = \frac{1}{2^k}\left[f_0(z)f_k(z) + \sum_{j=1}^{k}\frac{1}{(k-j)!}f_j(z)\right],\ k = 1, 2, \ldots, s. \tag{4.26}$$

The authors of [76] pointed out that the choice of the squaring laws is very important as generally this is the main source of errors committed in the algorithm (as will be explained later in this section) and concluded from their experiments that their choice (4.26) results in the minimum accumulation of errors.

In addition, the algorithm can also be based on the identity (taken from [37, 53])

$$f_1(2z) = \left(\frac{1}{2}zf_1(z) + 1\right)f_1(z), \tag{4.27}$$

and either the identities (4.21) - (4.22) or (4.24) - (4.25).

Note that we refer to the algorithm based on the identities (4.20) - (4.22) or (4.23) - (4.25) or (4.27) as the Scaling and Squaring algorithm.

Before we illustrate the algorithm, we verify the identities (4.20) - (4.22) and (4.27). For (4.20)

$$f_1(2z) = \frac{(e^z-1)(e^z+1)}{2z} = \frac{1}{2}f_1(z)\left(f_0(z)+1\right).$$

For (4.21)

$$\begin{aligned} f_2(2z) &= \frac{(e^z-1)(e^z+1)}{4z^2} - \frac{1}{2z}, \\ &= \frac{1}{4}f_1(z)\left(\frac{e^z+1}{z}\right) - \frac{1}{2z}, \\ &= \frac{1}{4}f_1(z)f_1(z) + \frac{1}{2z}\left(\frac{e^z-1-z}{z}\right), \\ &= \frac{1}{4}\left(f_1(z)f_1(z) + 2f_2(z)\right). \end{aligned}$$

And for (4.22)

$$\begin{aligned}
f_3(2z) &= \frac{(e^z-1)(e^z+1)}{8z^3} - \frac{1+z}{4z^2}, \\
&= \frac{1}{8}f_1(z)f_2(z) + \frac{(e^z-1)(2+z)}{8z^3} - \frac{1+z}{4z^2}, \\
&= \frac{1}{8}f_1(z)f_2(z) + \frac{e^z-1-z-z^2/2-z^2/2}{4z^3} + \frac{e^z-1}{8z^2}, \\
&= \frac{1}{8}f_1(z)f_2(z) + \frac{1}{4}f_3(z) + \frac{e^z-1-z}{8z^2}, \\
&= \frac{1}{8}\left(f_1(z)f_2(z) + f_2(z) + 2f_3(z)\right).
\end{aligned}$$

The identity (4.27) can be derived from the relation

$$f_1(2z) = \frac{e^z+1}{2}f_1(z),$$

and the relation

$$f_1(z) = \frac{e^z-1}{z} \Rightarrow e^z = zf_1(z)+1,$$

so that

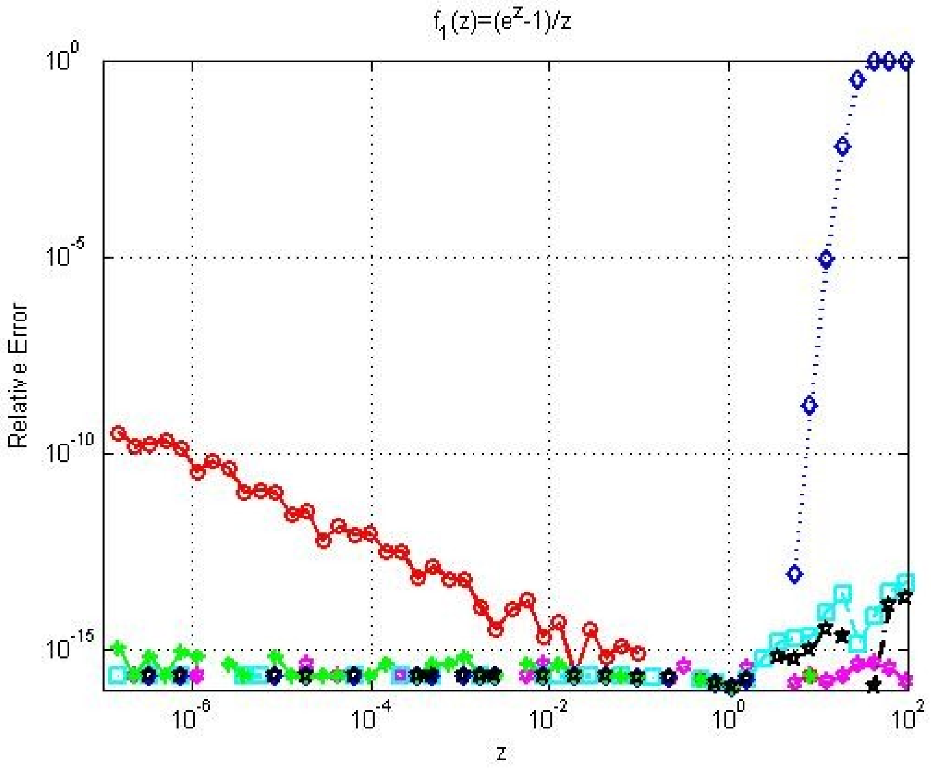
$$f_1(2z) = \frac{zf_1(z)+2}{2}f_1(z) = \Big(\frac{zf_1(z)}{2}+1\Big)f_1(z).$$

In implementing the Scaling and Squaring algorithm, we use a 30-term Taylor series to compute the expression $f_k(z)$ (4.2), $k = 1, 2, 3$ (as explained in §4.2.1) for values $|z| \le \delta$, for some threshold value δ. But for values $|z| > \delta$, the algorithm starts by the computation of $f_1(2^{-l}z)$, $f_2(2^{-l}z)$ and $f_3(2^{-l}z)$ for some l, again selected by the formula (4.18) so that the value of $|2^{-l}z| \le \delta$. For this evaluation we use a 30-term Taylor series[2]. We then proceed by applying the identities (4.20) - (4.22) or (4.23) - (4.25) (or (4.27) and either (4.21) - (4.22) or (4.24) - (4.25)) l times to compute the expression $f_k(z)$ (4.2) of orders $k = 1, 2, 3$, for the required values of z.

To demonstrate the algorithm's validity, we compute the relative error (4.10) of using this algorithm based on the identities (4.20) - (4.22) to approximate the expression $f_k(z)$ (4.2), $k = 1, 3$ (qualitatively similar results hold for $k = 2$) for values of z with small and large magnitude and with the choice of the threshold value $\delta = 1$. As displayed in figures 4.3 and 4.4, this algorithm is one of the most effective and powerful algorithms. It is stable for small positive values and for all negative values of z[3]. However, this algorithm is one of the most complex to

[2]Reasons for favoring the Taylor series than the Padé approximation are explained in §4.3.5.

[3]Qualitatively similar results hold when applying the identities (4.23) - (4.25) or (4.27) and either (4.21) - (4.22) or (4.24) - (4.25).



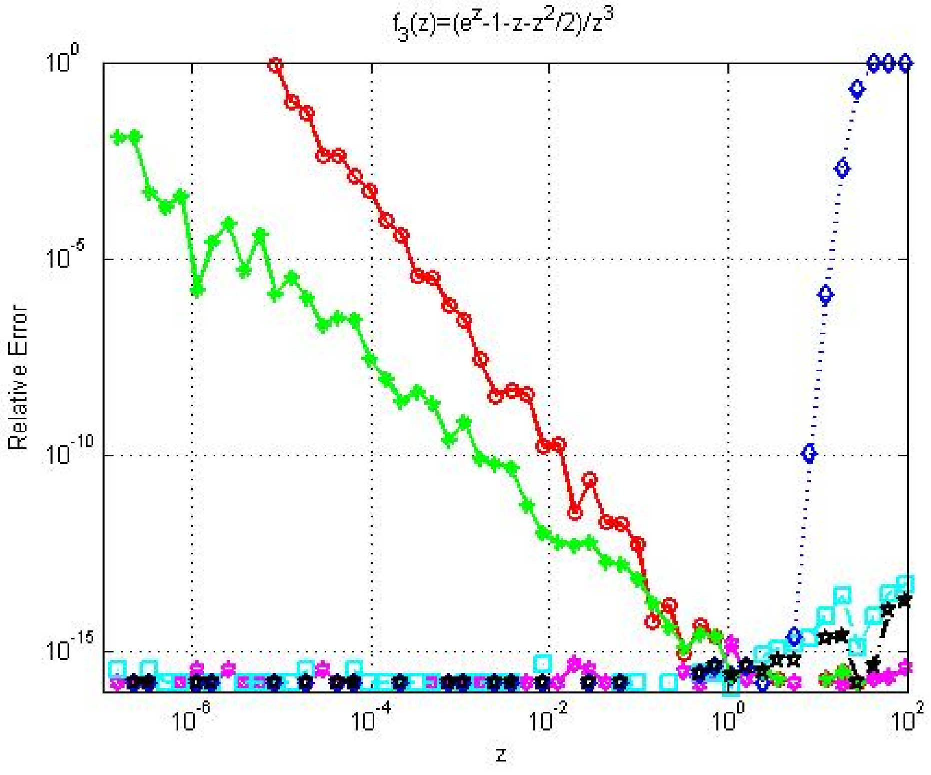
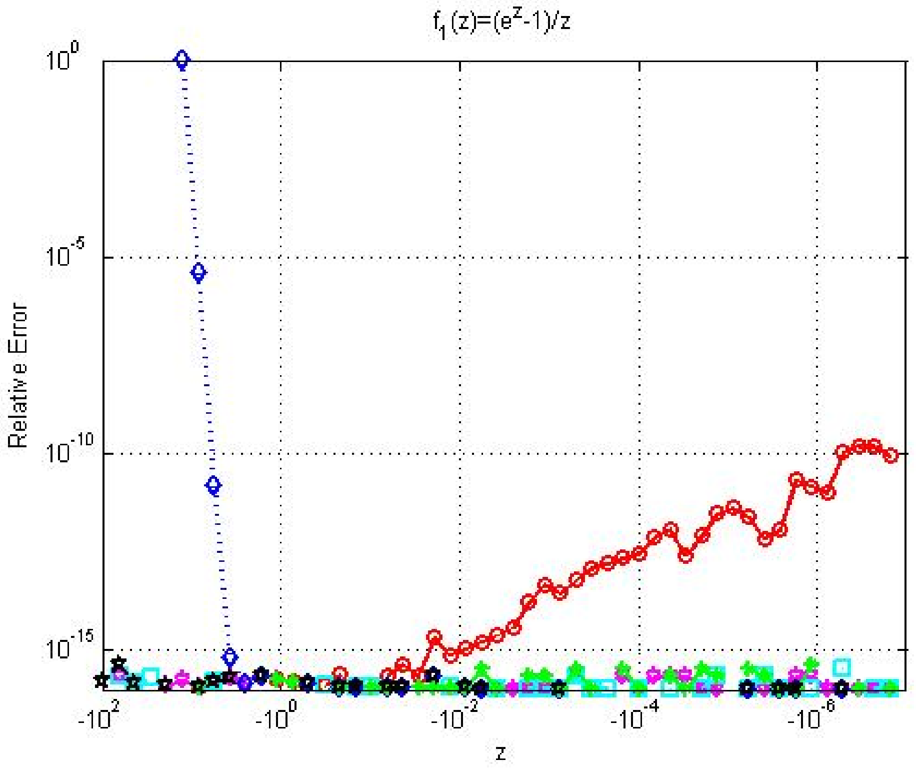


FIGURE 4.3: Relative errors in $f_1(z)$ (4.1) and $f_3(z)$ (4.5) versus the values $z > 0$ in the scalar case. The algorithms are: Explicit Formula (red circles), 30-term Taylor series (blue diamonds), the Cauchy Integral Formula (magenta stars), Scaling and Squaring Type **I** based on the identities (4.20) - (4.22) (black stars), Scaling and Squaring Type **II** based on the identities (4.48) - (4.50) (green circles) and Composite Matrix (cyan squares).



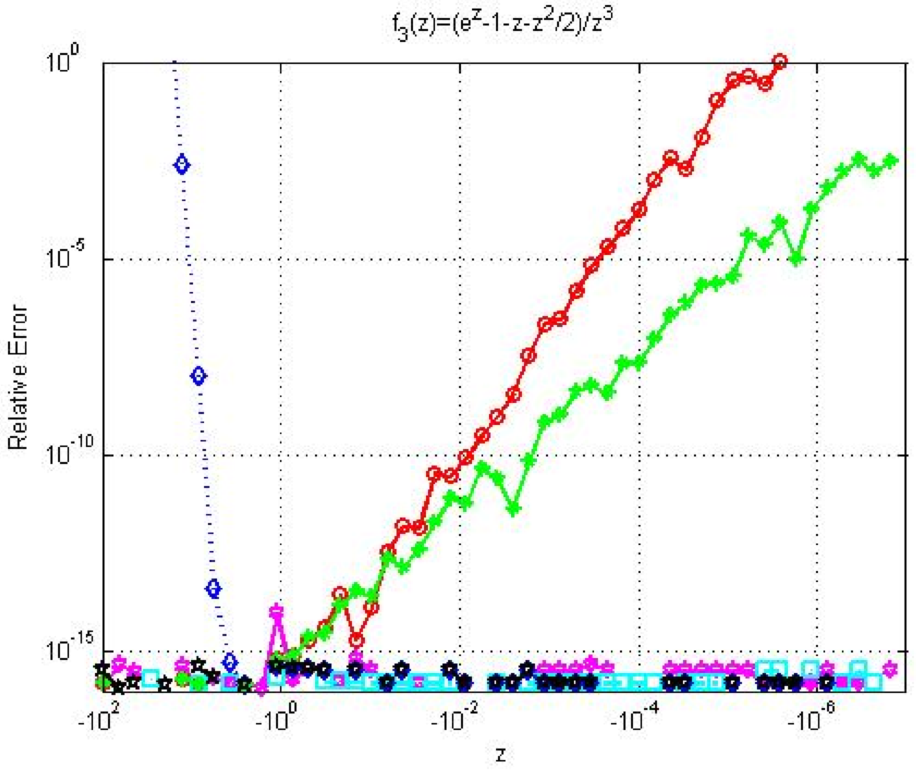


FIGURE 4.4: Relative errors in $f_1(z)$ (4.1) and $f_3(z)$ (4.5) versus the values $z < 0$ in the scalar case. The algorithms are: Explicit Formula (red circles), 30-term Taylor series (blue diamonds), the Cauchy Integral Formula (magenta stars), Scaling and Squaring Type **I** based on the identities (4.20) - (4.22) (black stars), Scaling and Squaring Type **II** based on the identities (4.48) - (4.50) (green circles) and Composite Matrix (cyan squares).

implement and its accuracy decreases as the value of $z > 1$ increases. This is due to the amplification of the truncation errors and the rounding errors (resulting from using the Taylor series) by the scaling and squaring process (these errors will be analyze shortly in this section).

It has been noted [35] that for a better performance of the algorithm, we should increase the threshold value δ as well as increasing the number of terms used in the Taylor series, so that the algorithm has fewer squarings to undo the effect of the scaling in approximating $f_0(z) = e^z$ using (4.19). These computed squares can be contaminated by rounding errors that are doubled at each scaling. To examine the effects of the squaring phase, in using the relation (4.19), on the approximated rounding errors, assume that the function $f_0(2^{-l}z)$ (for l selected by (4.18) so that $|2^{-l}z| \leq \delta$) is contaminated by some error ϵ in its computation and that the relative error is $\epsilon/|f_0(2^{-l}z)|$. Then, squaring $f_0(2^{-l}z)$ using the identity (4.19) l times to approximate $f_0(z)$ at $|z| \gg 1$ has rounding errors with

$$\textit{relative error} \approx \frac{|(f_0(2^{-l}z)+\epsilon)^{2^l} - f_0(2^{-l}z)^{2^l}|}{|f_0(2^{-l}z)^{2^l}|}. \tag{4.28}$$

Applying the binomial series

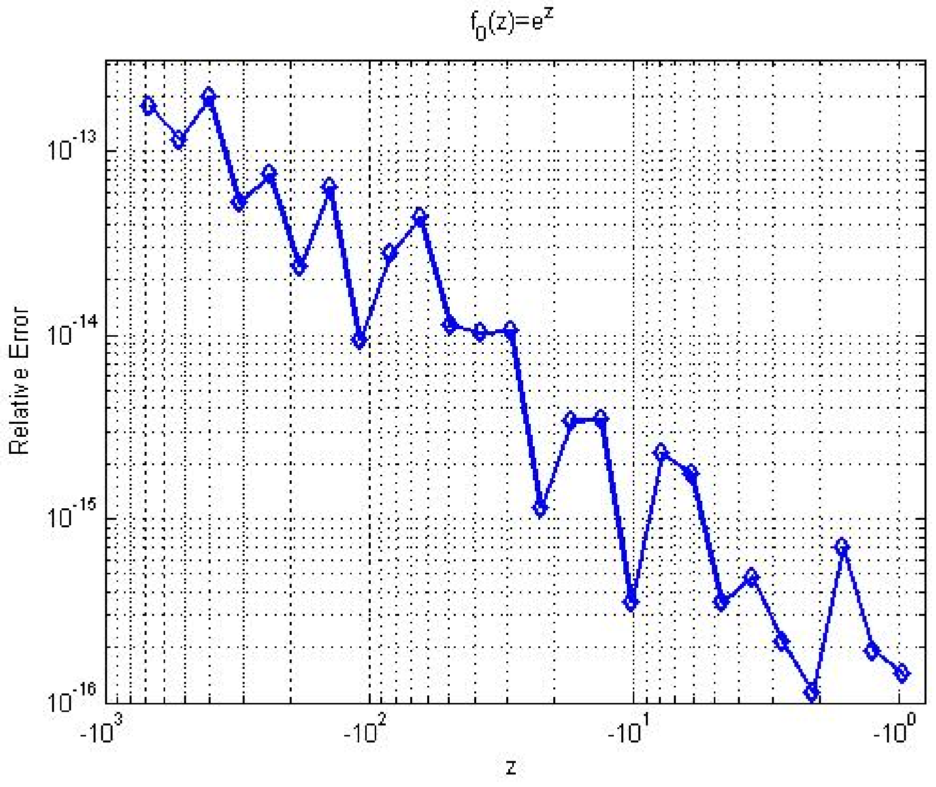
$$(x+y)^n = \sum_{j=0}^{n} \binom{n}{j} x^{n-j} y^j, \qquad \binom{n}{j} = \frac{n!}{j!(n-j)!}, \tag{4.29}$$

to the relative error (4.28) gives

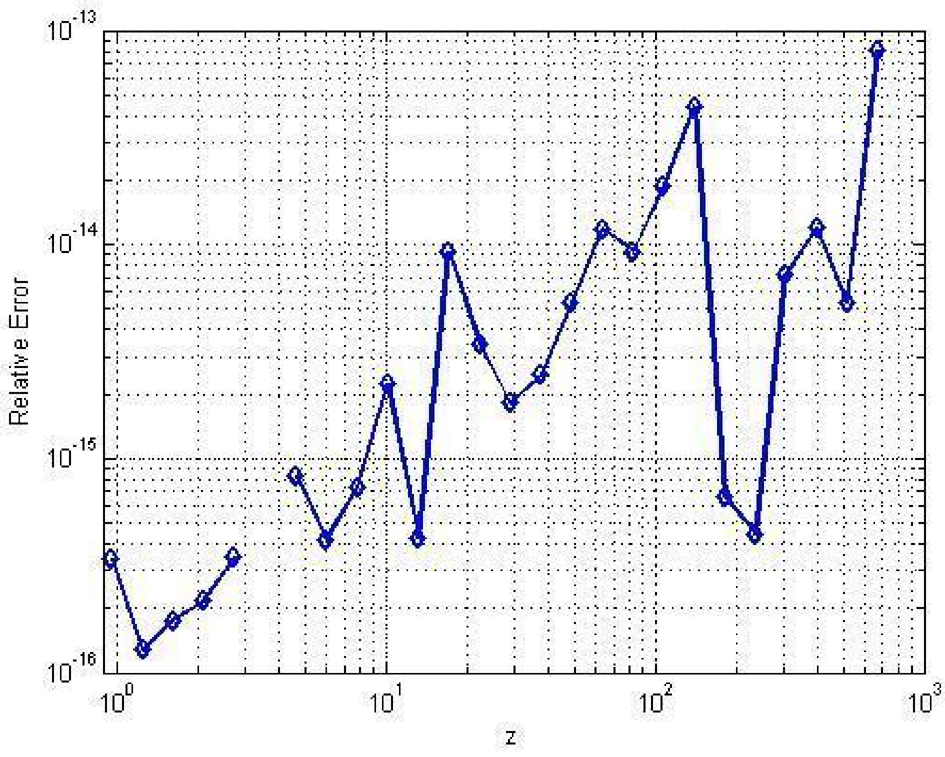
$$\begin{aligned} \textit{relative error} &\approx \frac{|2^l f_0(2^{-l}z)^{2^l-1}\epsilon + O(\epsilon^2)|}{|f_0(2^{-l}z)^{2^l}|}, \\ &\approx \frac{2^l \epsilon}{|f_0(2^{-l}z)|} \approx \frac{|z|\epsilon/\delta}{|f_0(2^{-l}z)|} \propto |z|, \end{aligned} \tag{4.30}$$

which shows that the errors are doubled at each scaling and we expect the relative error to increase linearly, by a factor of $2^l = |z|/\delta$ (see formula (4.18)), as $|z|$ increases. Figure 4.5 confirms the above analysis and illustrates the linear increase of the computed relative errors (4.30) of using (4.19) to approximate $f_0(z) = e^z$ for $|z| \gg 1$ with threshold value $\delta = 1$. Hence, it seems desirable to minimize the number of squarings in the algorithm.

For more analysis of this algorithm, the reader is referred to a paper by **Higham** [35] who gave a backward error analysis (in exact arithmetic) of the algorithm combined with Padé approximation (for computing the matrix exponential) that employs sharp bounds for the truncation errors in the approximant, and identified that



(a) $z < 0$



(b) $z > 0$

FIGURE 4.5: Relative errors of using the Scaling and Squaring Type **I** algorithm based on the identity (4.19), versus the values (a) $z < 0$ and (b) $z > 0$, for approximating the function $f_0(z)$ in the scalar case.

the loss of accuracy in the computed results is related to the number of squaring steps used and that larger values of the threshold may be optimal for the algorithm's optimal efficiency.

By looking at the relations (4.20) - (4.22) and (4.23) - (4.25), we find that they involve $f_0(z) = e^z$, which for its approximation depends on the scaling and squaring process using the identity (4.19). Therefore, the errors (4.30) in the squaring process will directly affect the use of these relations for approximating $f_k(z)$ (4.2), $k = 1, 2, 3$ at large positive values of z. So if $z \gg 1$, then $e^z \gg 1$ and therefore according to (4.6) the identities (4.20) - (4.22) become

$$f_1(2z) \approx \frac{1}{2} f_0(z) f_1(z), \tag{4.31}$$
$$f_2(2z) \approx \frac{1}{4} f_1(z) f_1(z), \tag{4.32}$$
$$f_3(2z) \approx \frac{1}{8} f_1(z) f_2(z), \tag{4.33}$$

respectively, and (4.24) - (4.25) become

$$f_2(2z) \approx \frac{1}{4} f_0(z) f_2(z), \tag{4.34}$$
$$f_3(2z) \approx \frac{1}{8} f_0(z) f_3(z), \tag{4.35}$$

respectively. This shows that applying the above identities to compute $f_k(z)$ (4.2), $k = 1, 2, 3$ for the required large positive value of z, will be affected by the error (4.30). But if $z \ll -1$ then $e^z \ll 1$ and therefore according to (4.8) the identities (4.20) - (4.22) become

$$f_1(2z) \approx \frac{1}{2} f_1(z), \tag{4.36}$$
$$f_2(2z) \approx \frac{1}{2} f_2(z), \tag{4.37}$$
$$f_3(2z) \approx \frac{1}{8} f_2(z) + \frac{1}{4} f_3(z), \tag{4.38}$$

respectively, and (4.24) - (4.25) become

$$f_2(2z) \approx \frac{1}{4}(f_1(z) + f_2(z)), \tag{4.39}$$
$$f_3(2z) \approx \frac{1}{8}(\frac{1}{2} f_1(z) + f_2(z) + f_3(z)), \tag{4.40}$$

respectively. Hence, the identities (4.36) - (4.40) do not involve $f_0(z) = e^z$, the error (4.30) in applying the identity (4.19) has no effect on these identities when they are applied for all required values $z < 0$, and thus the algorithm becomes more stable and accurate.

The above analysis explains the behavior of the algorithm based on the identities (4.20) - (4.22), displayed in figures 4.3 and 4.4, where the errors grow for $z \gg 1$ but not for $z \ll -1$.

To analyze the rounding errors resulting from applying the identity (4.27)

$$f_1(2z) = \Big(\frac{1}{2} z f_1(z) + 1\Big) f_1(z),$$

to compute $f_1(z)$ (4.1), assume that the exact value of $f_1(z)$ is contaminated by some error ϵ_1 in its computation so that

$$relative\ error = \frac{\epsilon_1}{|f_1(z)|}. \tag{4.41}$$

Applying the identity (4.27) then has rounding errors with

$$\begin{aligned} relative\ error &\approx \frac{|f_{1_{approx}}(2z) - f_{1_{exact}}(2z)|}{|f_{1_{exact}}(2z)|}, \\ &\approx \frac{|f_1(z) z \epsilon_1/2 + \epsilon_1 z f_1(z)/2 + \epsilon_1|}{|f_1(z) z f_1(z)/2 + f_1(z)|}. \end{aligned} \tag{4.42}$$

As $z \to \infty$, $e^z \gg 1$, $f_1(z) \approx e^z/z$, the relative error (4.41) becomes

$$relative\ error \approx \frac{\epsilon_1 z}{e^z}, \tag{4.43}$$

and (4.42) becomes

$$\begin{aligned} relative\ error &\approx \frac{|e^z \epsilon_1/2 + \epsilon_1 e^z/2 + \epsilon_1|}{|e^{2z}/2z + e^z/z|}, \\ &\approx \frac{2\epsilon_1 z}{e^z}, \end{aligned} \tag{4.44}$$

which shows that the errors (4.43) are doubled at each scaling and that the algorithm becomes less accurate as the value of positive z increases. In this case, the identities (4.32) and (4.33) will also be affected by the error (4.44) when they are applied to compute $f_2(z)$ (4.4) and $f_3(z)$ (4.5).

On the other hand, for values of $z \to -\infty$, $e^z \ll 1$ and $f_1(z) \approx -1/z$ and therefore the order ϵ_1 terms in (4.42) simplify to

$$relative\ error \approx 0,$$

which shows the algorithm's validity when applying the identities (4.27) (and when subsequently applying either identities (4.21) - (4.22) or (4.24) - (4.25) for approximating $f_k(z)$ (4.2), $k = 1, 2, 3$ for all values $z < 0$).

In further experiments on the Scaling and Squaring algorithm, we investigate what the best choice of the threshold value δ is, among certain chosen values $\delta =$

$0.5, 1, 2, 3$. In figure 4.6, and for each chosen value of the threshold δ, we plot the computed relative errors (4.10) of using the algorithm, based on the identities (4.20) - (4.22), to approximate $f_3(z)$ (4.5) (qualitatively similar results hold for the expressions $f_1(z)$ (4.1) and $f_2(z)$ (4.4)) for positive values of z[4]. The figure reveals that, the choices of threshold values $\delta = 0.5, 1, 2$ are all good, giving better results than the value $\delta = 3$. At larger value $\delta \geq 3$, we experimentally find that increasing the value of the threshold requires an increase in the number of terms used in the Taylor series combined with the algorithm for better accuracy.

In addition, in figure 4.7, we note that the computed relative errors (4.10) of using the Scaling and Squaring algorithm to approximate $f_3(z)$ (qualitatively similar results hold for the expressions $f_1(z)$ and $f_2(z)$) for positive values of z are, firstly, similar regardless of which relations (the identities (4.20) - (4.22) or (4.23) - (4.25) or (4.27) and either (4.21) - (4.22) or (4.24) - (4.25)) are used and whatever the chosen value of the threshold δ is (in the figure $\delta = 1$). Secondly, these errors increase linearly for $z \gg 1$, which agrees with the above analysis. On the other hand, in doing the same experiment for negative values of z (figures are not shown), we find that the errors are smaller (errors of $O(10^{-15})$) and they do not grow linearly for values of $z \to -\infty$ as our above analysis suggests.

4.2.4 Scaling and Squaring Algorithm: Type II

Recall that the numerical evaluation of the explicit formula $f_k(z)$ (4.2), $k = 1, 3$ is accurate for scalar values $|z| > 1$ but not for $|z| < 1$ (the same qualitatively holds for $k = 2$), see figures 4.3 and 4.4. This suggests a second type of Scaling and Squaring algorithm, based on scaling down from $|z| > 1$.

Consider again the evaluation of the exponential function. For values of $|z| \geq \gamma$, for some threshold value γ, we use the function $f_0(z) = e^z$, but for values $|z| < \gamma$, we use the Scaling and Squaring algorithm based on the identity, which we now write in the form

$$f_0(z) = (f_0(2z))^{1/2} = (e^{2z})^{1/2}. \tag{4.45}$$

First we compute $f_0(2^{l_1} z)$ using the exponential function for some l_1 chosen to be

[4]Qualitatively similar results hold when the algorithm is based on the identities (4.23) - (4.25) or (4.27) and either (4.21) - (4.22) or (4.24) - (4.25) and for negative values of z.

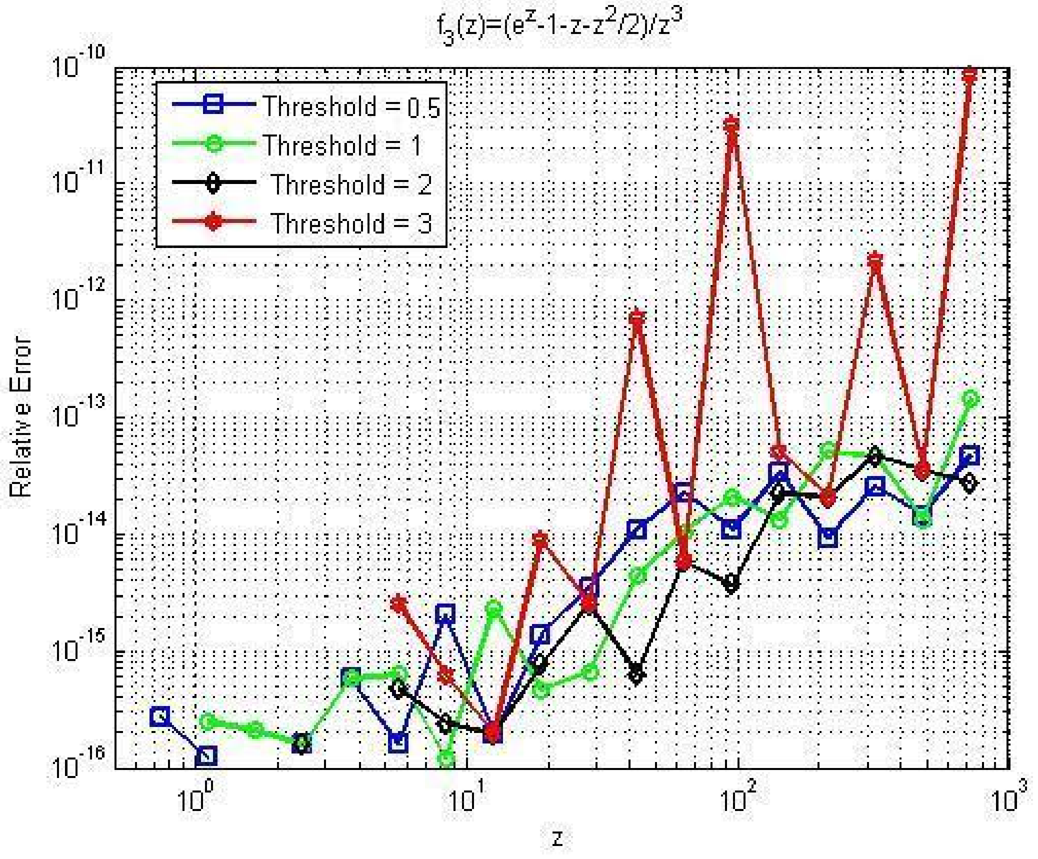


Figure 4.6: Relative errors of using the Scaling and Squaring Type **I** algorithm based on the identities (4.20) - (4.22), versus the values of z, for approximating the expression $f_3(z)$ (4.5), for different values of threshold δ (see formula (4.18)).

the smallest integer such that

$$l_1 \geq \frac{\log(\gamma/|z|)}{\log 2}, \tag{4.46}$$

so that the value of $|2^{l_1}z| \geq \gamma$. Using (4.45) the resulting value is then square-rooted l_1 times to obtain the final answer

$$f_0(z) = [f_0(2^{l_1}z)]^{1/2^{l_1}}. \tag{4.47}$$

A similar approach can be used for computing the expression $f_k(z)$ (4.2) of orders $k = 1, 2, 3$. For values of z with large or moderate magnitude we can simply use the formula $f_k(z)$ (4.2), which give accurate results, but for values of z with small magnitude we use either the identities

$$f_1(z) = 2f_1(2z)/(f_0(z)+1), \tag{4.48}$$

$$f_2(z) = 2f_2(2z) - \frac{1}{2}f_1(z)f_1(z), \tag{4.49}$$

$$f_3(z) = 4f_3(2z) - \frac{1}{2}f_1(z)f_2(z) - \frac{1}{2}f_2(z), \tag{4.50}$$

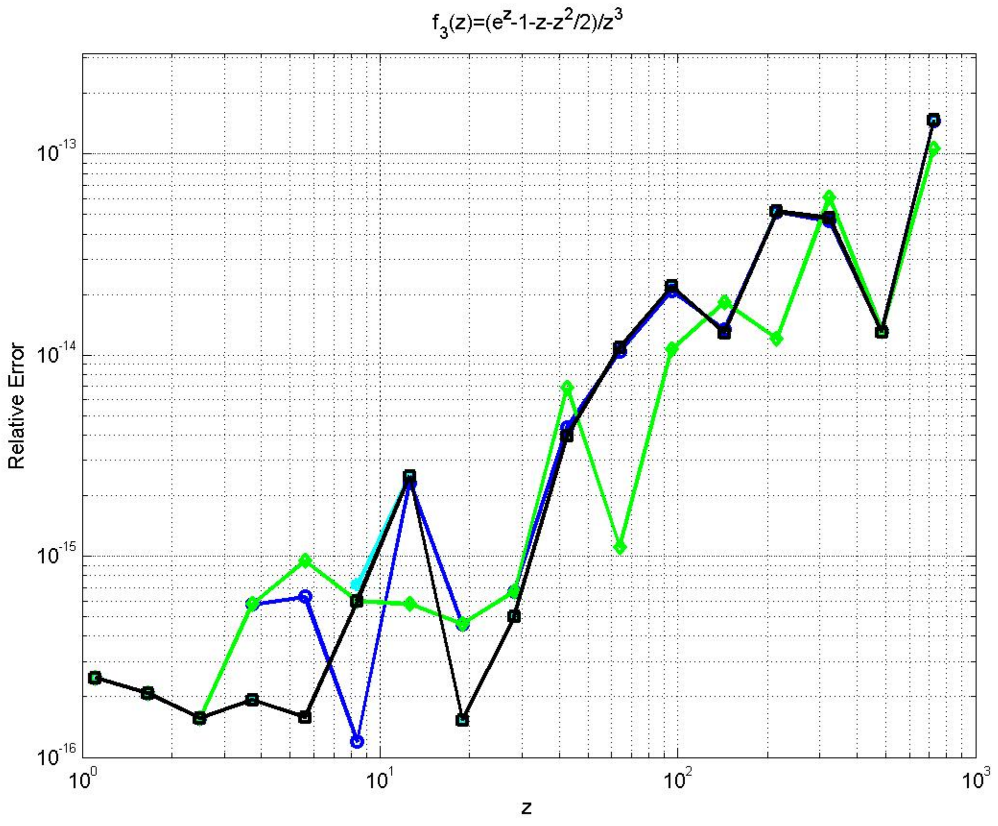


FIGURE 4.7: Relative errors of using the Scaling and Squaring Type **I** algorithm, versus the values of z, for approximating the expression $f_3(z)$ (4.5). The blue line (circles) uses the identities (4.20) - (4.22), the cyan line (stars) uses the identities (4.23) - (4.25), the green line (diamonds) uses the identities (4.27), (4.21) and (4.22) and the black line (squares) uses the identities (4.27), (4.24) and (4.25).

or

$$f_1(z) = 2f_1(2z)/(f_0(z)+1), \tag{4.51}$$

$$f_2(z) = (4f_2(2z) - f_1(z))/(f_0(z)+1), \tag{4.52}$$

$$f_3(z) = (8f_3(2z) - \frac{1}{2}f_1(z) - f_2(z))/(f_0(z)+1). \tag{4.53}$$

The identities (4.48) - (4.50) and (4.51) - (4.53) are formed by rearranging the identities (4.20) - (4.22) and (4.23) - (4.25) respectively.

We start by computing $f_1(2^{l_1}z)$, $f_2(2^{l_1}z)$ and $f_3(2^{l_1}z)$ using the formula $f_k(z)$ (4.2) for $k = 1,2,3$ respectively, which will be accurate, for some l_1 selected by the formula (4.46), so that the value of $|2^{l_1}z| \geq \gamma$, which we choose here to be $\gamma = 1$. The identities (4.48) - (4.50) or (4.51) - (4.53) are then applied l_1 times to compute the expression $f_k(z)$ (4.2) of orders $k = 1,2,3$ for the required values of z.

To examine the effects of using (4.47) to compute $f_0(z) = e^z$ on rounding errors, assume that the function $f_0(2^{l_1}z)$, for l_1 selected by the formula (4.46), is contaminated by some error ϵ in its computation so that the relative error is $\epsilon/|f_0(2^{l_1}z)|$. Taking the square-root of $f_0(2^{l_1}z)$ l_1 times, it follows that using the identity (4.47), to approximate $f_0(z)$ at $|z| \ll 1$ has rounding errors with

$$relative\ error \approx \frac{|(f_0(2^{l_1}z)+\epsilon)^{2^{-l_1}} - f_0(2^{l_1}z)^{2^{-l_1}}|}{|f_0(2^{l_1}z)^{2^{-l_1}}|}. \tag{4.54}$$

Applying the binomial series (4.29) to the relative error (4.54) gives

$$\begin{aligned} relative\ error &\approx \frac{|2^{-l_1}f_0(2^{l_1}z)^{2^{-l_1}-1}\epsilon + O(\epsilon^2)|}{|f_0(2^{l_1}z)^{2^{-l_1}}|}, \\ &\approx \frac{2^{-l_1}\epsilon}{|f_0(2^{l_1}z)|} \approx \frac{|z|\epsilon/\gamma}{|f_0(2^{l_1}z)|} \propto |z|, \end{aligned} \tag{4.55}$$

which shows that the errors are halved at each scaling and we expect the relative error to decrease linearly with $|z|$, by a factor of 2^{l_1}, as $|z|$ is halved l_1 times.

We may carry out a similar analysis to analyze the rounding errors resulting from applying the identity (4.48)

$$f_1(z) = \frac{2f_1(2z)}{f_0(z)+1},$$

to compute the function $f_1(z)$ (4.1) for the required values of z. To do this, we first assume that errors in approximating $f_0(z)$ by applying (4.47) are negligible, due to the result (4.55), and that the exact value of the function $f_1(2z)$ is contaminated by some error ϵ_1, with relative error $\epsilon_1/|f_1(2z)|$. Thus, applying the identity (4.48) has rounding errors with

$$\begin{aligned} relative\ error &\approx \frac{|f_{1_{approx}}(z) - f_{1_{exact}}(z)|}{|f_{1_{exact}}(z)|}, \\ &\approx \Big|\frac{2f_1(2z)+2\epsilon_1-2f_1(2z)}{f_0(z)+1}\Big| \Big/ \Big|\frac{2f_1(2z)}{f_0(z)+1}\Big|, \\ &\approx \frac{\epsilon_1}{|f_1(2z)|}. \end{aligned} \tag{4.56}$$

This shows that there is no amplification of the errors at each scaling and that the algorithm's accuracy remains the same. For $|z| \ll 1$ and according to (4.7), $f_1(2z) \approx 1$ and the relative error (4.56) becomes

$$relative\ error \approx \epsilon_1. \tag{4.57}$$

When applying the above ideas to analyze the rounding errors resulting from applying the identities (4.49) and (4.48) to compute $f_2(z)$ (4.4), we now assume that the errors in applying the identity (4.48) are negligible, because these errors are not amplified (not growing) according to (4.57). Then if the relative error in approximating $f_2(2z)$ is

$$\textit{relative error} = \frac{\epsilon_2}{|f_2(2z)|}, \tag{4.58}$$

for some error ϵ_2, the relative error in applying the identity $f_2(z)$ (4.49) is

$$\textit{relative error} \approx \frac{2\epsilon_2}{|2f_2(2z) - \frac{1}{2}f_1(z)f_1(z)|}. \tag{4.59}$$

As $z \to 0^{\pm}$ and according to (4.7), $f_1(z) \approx 1$, $f_2(2z) \approx 1/2$, the relative error (4.58) becomes

$$\textit{relative error} \approx 2\epsilon_2, \tag{4.60}$$

and (4.59) becomes

$$\textit{relative error} \approx 4\epsilon_2,$$

which shows that errors (4.60) are doubled at each scaling and we expect the relative error to increase linearly, by a factor of 2^{l_1}, as $|z|$ is halved l_1 times, i.e. the *relative error* $\propto 1/|z|$.

As above, we may analyze the rounding errors resulting from applying the identities (4.48) - (4.50), assuming again that the errors in applying (4.48) are negligible, because these errors are not amplified according to (4.57). If the relative error in approximating $f_3(2z)$ is

$$\textit{relative error} = \frac{\epsilon_3}{|f_3(2z)|}, \tag{4.61}$$

for some error ϵ_3, then the relative error in applying the identity (4.50) is

$$\textit{relative error} \approx \frac{4\epsilon_3 - \epsilon_2(f_1(z)+1)/2}{|4f_3(2z) - \frac{1}{2}f_1(z)f_2(z) - \frac{1}{2}f_2(z)|}. \tag{4.62}$$

Since the relative error in approximating $f_3(z)$ is growing faster by a factor of 2 than that of approximating $f_2(z)$ (due to (4.59) and (4.62)), we assume that the relative error of approximating $f_2(z)$ is small compared to that of approximating $f_3(z)$ and therefore it can be ignored, and so (4.62) becomes

$$\textit{relative error} \approx \frac{4\epsilon_3}{|4f_3(2z) - \frac{1}{2}f_1(z)f_2(z) - \frac{1}{2}f_2(z)|}. \tag{4.63}$$

As $z \to 0^{\pm}$ and according to (4.7), $f_1(z) \approx 1$, $f_2(z) \approx 1/2$, $f_3(2z) \approx 1/6$, the relative error (4.61) becomes

$$\textit{relative error} \approx 6\epsilon_3, \tag{4.64}$$

and (4.63) becomes

$$\textit{relative error} \approx 24\epsilon_3,$$

which shows that errors (4.64) are amplified by a factor of 4 at each scaling, and we expect the relative error to increase by a factor of $(2^{l_1})^2$ as $|z|$ is halved l_1 times, i.e. the *relative error* $\propto 1/z^2$.

Regarding the test described in §4.2, and according to figures 4.3 and 4.4, the Scaling and Squaring algorithm based on the identity (4.48) performs well overall when evaluating the simplest expression $f_1(z)$ (4.1). But when numerically computing the relative error (4.10) of applying the identities (4.48) - (4.50)[5] to compute the function $f_3(z)$ (4.5) for values of z with small magnitude, we find that the results are inaccurate[6]. The results of using the algorithm shown in figures 4.3 and 4.4 agree well with the above analysis, and thus, for values of $z \to 0^{\pm}$, the Scaling and Squaring type **II** algorithm is not a very stable nor a useful algorithm.

4.2.5 Composite Matrix Algorithm

Although this algorithm is not explicitly given in earlier work, related algorithms appear in [2, 56, 67, 73]. The algorithm starts with the construction of an $(s+1)\times(s+1)$ matrix with the structure

$$B1_s = \begin{pmatrix} z & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 \end{pmatrix}. \tag{4.65}$$

If we exponentiate the matrix $B1_s$, the resulting matrix is

$$e^{B1_s} = \begin{pmatrix} e^z & f_1(z) & f_2(z) & f_3(z) & f_4(z) & \cdots & f_s(z) \\ 0 & 1 & 1 & 1/2 & 1/3! & \cdots & 1/(s-1)! \\ 0 & 0 & 1 & 1 & 1/2 & \cdots & 1/(s-2)! \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}, \tag{4.66}$$

[5]Qualitatively similar results are found when using the identities (4.51) - (4.53).

[6]Qualitatively similar results are found when approximating the function $f_2(z)$ (4.4).

which can be verified directly using the Taylor series expansion of the exponential function. We note in particular that, due to the structure of $B1_s$, any power of the matrix $B1_s$ contains as an element the corresponding power of the value z in the same position where $B1_s$ contains z, and therefore, the exponential of z will be generated in the same position.

To prove the result (4.66), we start by exponentiating the matrix $B1_s$ (4.65) using the Taylor series expansion which gives

$$\begin{aligned} e^{B1_s} &= \sum_{n=0}^{\infty} \frac{B1_s^n}{n!}, \\ &= I + B1_s + B1_s^2/2! + B1_s^3/3! + B1_s^4/4! + \ldots . \end{aligned} \tag{4.67}$$

Note that $B1_s^0 = I$ and

$$B1_s^n = \begin{cases} \begin{pmatrix} z^n & z^{n-1} & \cdots & z^0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 1_{(s-n+1)\times(s+1)} \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \end{pmatrix}, & \text{if } n < s \\ \begin{pmatrix} z^n & z^{n-1} & z^{n-2} & z^{n-3} & z^{n-4} & z^{n-5} & \cdots & z^{n-s} \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \end{pmatrix}, & \text{if } n \geq s. \end{cases} \tag{4.68}$$

Therefore, using (4.68) we can rewrite (4.67) as follows

$$e^{B1_s} = \begin{pmatrix} \sum\limits_{n=0}^{s-1}\frac{z^n}{n!} & \sum\limits_{n=1}^{s-1}\frac{z^{n-1}}{n!} & \sum\limits_{n=2}^{s-1}\frac{z^{n-2}}{n!} & \sum\limits_{n=3}^{s-1}\frac{z^{n-3}}{n!} & \cdots & \sum\limits_{n=s-2}^{s-1}\frac{z^{n-s+2}}{n!} & \sum\limits_{n=s-1}^{s-1}\frac{z^{n-s+1}}{n!} & 0 \\ 0 & 1 & 1/1! & 1/2! & \cdots & 1/(s-3)! & 1/(s-2)! & 1/(s-1)! \\ 0 & 0 & 1 & 1/1! & \cdots & 1/(s-4)! & 1/(s-3)! & 1/(s-2)! \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1/1! & 1/2! \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 1/1! \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}$$

$$+ \begin{pmatrix} \sum\limits_{n=s}^{\infty}\frac{z^n}{n!} & \sum\limits_{n=s}^{\infty}\frac{z^{n-1}}{n!} & \sum\limits_{n=s}^{\infty}\frac{z^{n-2}}{n!} & \sum\limits_{n=s}^{\infty}\frac{z^{n-3}}{n!} & \cdots & \sum\limits_{n=s}^{\infty}\frac{z^{n-s+2}}{n!} & \sum\limits_{n=s}^{\infty}\frac{z^{n-s+1}}{n!} & \sum\limits_{n=s}^{\infty}\frac{z^{n-s}}{n!} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{pmatrix}.$$

The first matrix arises from the sum of the terms in the series (4.67) for which $n < s$ and the second arises from the terms in the sum with $n \geq s$. Adding together these two matrices gives the result of

$$e^{B1_s} = \begin{pmatrix} \sum\limits_{n=0}^{\infty}\frac{z^n}{n!} & \sum\limits_{n=1}^{\infty}\frac{z^{n-1}}{n!} & \sum\limits_{n=2}^{\infty}\frac{z^{n-2}}{n!} & \sum\limits_{n=3}^{\infty}\frac{z^{n-3}}{n!} & \cdots & \sum\limits_{n=s-2}^{\infty}\frac{z^{n-s+2}}{n!} & \sum\limits_{n=s-1}^{\infty}\frac{z^{n-s+1}}{n!} & \sum\limits_{n=s}^{\infty}\frac{z^{n-s}}{n!} \\ 0 & 1 & 1/1! & 1/2! & \cdots & 1/(s-3)! & 1/(s-2)! & 1/(s-1)! \\ 0 & 0 & 1 & 1/1! & \cdots & 1/(s-4)! & 1/(s-3)! & 1/(s-2)! \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1/1! & 1/2! \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 1/1! \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}.$$

If we consider expanding the expression $f_k(z)$ (4.2), $k = 1, 2, \ldots, s$ using the Taylor series expansion as in equation (4.9), then we have completed the proof of

(4.66), that is

$$e^{B1_s} = \begin{pmatrix} e^z & f_1(z) & f_2(z) & f_3(z) & \cdots & f_{s-2}(z) & f_{s-1}(z) & f_s(z) \\ 0 & 1 & 1/1! & 1/2! & \cdots & 1/(s-3)! & 1/(s-2)! & 1/(s-1)! \\ 0 & 0 & 1 & 1/1! & \cdots & 1/(s-4)! & 1/(s-3)! & 1/(s-2)! \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1/1! & 1/2! \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 1/1! \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}.$$

This algorithm evaluates the expression $f_k(z)$ (4.2) of orders $k = 1, 2, \ldots, s$, which are contained in the matrix (4.66) and can be extracted easily, assuming that we have a reliable function for computing the matrix exponential (such as the Matlab function *expm*, which uses a scaling and squaring method combined with Padé approximation (4.94) [35, 37, 47, 76]).

This algorithm is very attractive, being very simple and easily programmed. The approximations of the expressions $f_1(z)$ (4.1) and $f_3(z)$ (4.5) for small positive values of z, shown in figure 4.3, and for all values $z < 0$, shown in figure 4.4, are accurate to within machine precision (qualitatively similar results are found for the expression $f_2(z)$ (4.4)). As the value of positive z increases, the performance of the algorithm deteriorates, see figure 4.3. This is due to the increase in the norm of the matrix $B1_s$ (4.65), which leads to an increase in the number of scalings needed to approximate the matrix exponential e^{B1_s} (4.66). This scaling and squaring process amplifies the truncation errors and the rounding errors resulting from the matrix inversion and the repeated matrix multiplications when using the Padé approximation, see §4.3.5. In fact these errors are doubled at each scaling, as shown in (4.30), and we expect the relative error to increase linearly as the value of positive z increases (see in §4.2.3 the analysis of the rounding errors in using the Scaling and Squaring algorithm for approximating the exponential function).

4.3 Non-Diagonal Matrix Case

Implementing the ETD methods [19] as a time discretizing method for a system of ODEs, where the linear operator is represented by a non-diagonal matrix, requires the computation of matrix functions that involve the matrix exponential.

As discussed at the start of the chapter, in addition to the difficulties inherent in computing the matrix exponential itself, accurate evaluation of the matrix functions can be problematic when the matrix has small eigenvalues. This is a well known problem in numerical analysis. Various algorithms have been proposed by many authors [2, 8, 35, 47, 54, 56, 57, 67, 80, 81], and have been investigated in terms of their practical efficiency. For example, **Schmelzer** and **Trefethen** [69, 70] discussed the efficient computation of matrix functions. They proposed two methods for the fast evaluation of these functions building on previous work by **Trefethen** and **Gutknecht, Minchev**, and **Lu**. The first method is based on computing optimal rational approximations to the matrix functions on the negative real axis using the **Carathéodory-Fejér** procedure [85]. The second method is an application of the Trapezium rule on a Talbot-type contour encircling the eigenvalues of the matrix.

Computing the matrix exponential alone has also attracted several authors' attention. For example, **Beylkin** et al. [9] used the algorithm that is based on scaling and squaring to approximate a matrix exponential. Also, following an original paper on this problem [59], **Moler** and **Van Loan** [60] recently revisited this problem in "Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later", in which they described recent developments in computing the exponential of a matrix, and provided some interesting analysis and applications of some of the algorithms mentioned previously in this chapter. They cautioned that practical implementations are 'dubious' in the sense that implementation of a sole algorithm might not be entirely reliable for all classes of problems.

To investigate the algorithms' performance in the non-diagonal matrix case, we set up a large number of computational experiments on various orders q of the second-order centered difference differentiation matrix (see §2.2) for the second derivative,

$$M_2 = \begin{pmatrix} -2 & 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & -2 \end{pmatrix}, \tag{4.69}$$

(note that if the order of the matrix M_2 (4.69) is q, the scaling of M_2 is such that it corresponds to the second derivative on an interval of length $q+1$).

Tests on the Chebyshev differentiation matrix for the second derivative [11, 25, 83, 84] and the second-order centered difference differentiation matrix for the first derivative are described in §4.4 and §4.5 respectively.

We use the Matlab function *expm* to approximate the exponential function $e^{\Delta tM}$ of a matrix M, and the function *inv* to find $(\Delta tM)^{-1}$, and 50 digit arithmetic to approximate the exact values of the expressions

$$f_1(\Delta tM) = \frac{e^{\Delta tM} - I}{\Delta tM}, \tag{4.70}$$

$$f_2(\Delta tM) = \frac{e^{\Delta tM} - I - \Delta tM}{(\Delta tM)^2}, \tag{4.71}$$

and

$$f_3(\Delta tM) = \frac{e^{\Delta tM} - I - \Delta tM - (\Delta tM)^2/2}{(\Delta tM)^3}, \tag{4.72}$$

where I is the $q \times q$ identity matrix and Δt is the time step, that are required for the ETD1 (3.14) and the ETD2 (3.15) methods, respectively, in the matrix case. For the ETD3 (3.16) and higher order methods (also the ETD-RK methods), the coefficients are really a combination of the expression

$$f_k(\Delta tM) = \frac{e^{\Delta tM} - G_k(\Delta tM)}{(\Delta tM)^k}, \quad k = 1, 2, \ldots, s, \tag{4.73}$$

where

$$G_k(\Delta tM) = \sum_{j=0}^{k-1} \frac{(\Delta tM)^j}{j!}, \tag{4.74}$$

is the first k terms in the Taylor series approximation to the exponential function $f_0(\Delta tM) = e^{\Delta tM}$ and $(\Delta tM)^0 = I$. These coefficients can be evaluated using the algorithms (to be explained later in this section), in a manner similar to evaluating the expression $f_k(\Delta tM)$ (4.73), $k = 1, 2, \ldots, s$ by those algorithms.

The definition of the 2−norm of a matrix [78]

$$||\Delta tM||_2 = max \frac{||\Delta tMx||_2}{||x||_2},$$

where $x \in \mathbb{R}^q_* = \mathbb{R}^q \backslash \{0\}$, is equivalent to the formula

$$||\Delta tM||_2 = \sqrt{\zeta_{max}((\Delta tM)^T(\Delta tM))}, \tag{4.75}$$

(the square root of the maximum eigenvalue ζ_{max} of the matrix multiplied by its transpose). Formula (4.75) are used in our experiments to find the numerical relative errors (4.10) of using each algorithm to approximate the expression $f_k(\Delta tM)$ (4.73), $k = 1, 2, 3$ for large and small values of the time step Δt. In figure 4.8, we present only the results of our experiments for the expressions $f_2(\Delta t M_2)$ (4.71) and $f_3(\Delta t M_2)$ (4.72) in the 40×40 matrix case, since those for the expression $f_1(\Delta t M_2)$ (4.70) are qualitatively similar. The size of the matrix used is limited not by the time used by the algorithms but by the much greater time needed to obtain the 'exact' 50-digit results. Results for smaller and larger matrices are qualitatively similar. Note that figure 4.8 also shows the errors for the use of the explicit formulas; in the matrix case this means simply evaluating the formulas $f_2(\Delta t M_2)$ and $f_3(\Delta t M_2)$ using the Matlab commands *expm* and *inv* with standard double precision (16 digits) arithmetic (the function *expm* uses a scaling and squaring method combined with Padé approximation (4.94) [35, 37, 47, 76], and therefore is not quite explicit).

4.3.1 Taylor Series

The approximation

$$e^{\Delta t M} \approx I + \Delta t M + \frac{(\Delta t M)^2}{2!} + \frac{(\Delta t M)^3}{3!} + \cdots + \frac{(\Delta t M)^m}{m!},$$

where I is the $q \times q$ identity matrix, for some integer m may be used to approximate the exponential in the expression $f_k(\Delta t M)$ (4.73), so that

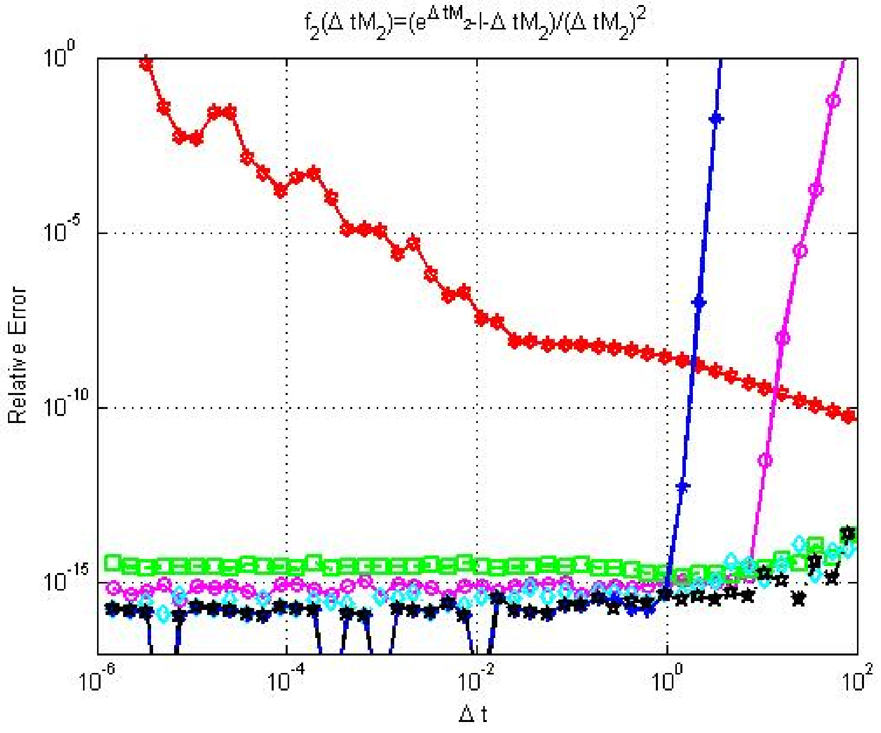
$$f_k(\Delta t M) \approx \sum_{j=k}^{m} \frac{(\Delta t M)^{j-k}}{j!}, \qquad k = 1, 2, 3, \tag{4.76}$$

where $(\Delta t M)^0 = I$. However, it is well known that although in principle this series is convergent, in practice the algorithm is very inaccurate when $||\Delta t M||$ is large (see, for example [60]).

The 30-term Taylor series algorithm is one of the easiest algorithms to implement in the matrix case. However, as expected, it does not perform very well for large values of Δt, as is indicated in figure 4.8. The problem in using the Taylor expansion directly is that it results in a loss of accuracy, because some of the eigenvalues of the $q \times q$ matrix $\Delta t M_2$ are negative and much less than -1 for large values of Δt. Therefore the problem of cancellation reappears (see §4.2.1). The eigenvalues λ_j of the matrix $\Delta t M_2$ (4.69) can be derived analytically (see [43]) in the form

$$\lambda_j = \left(-2 + 2\cos\left(\frac{j\pi}{q+1}\right)\right)\Delta t, \qquad j = 1, \cdots, q,$$

so the eigenvalue of largest magnitude is $\lambda_q \approx -4\Delta t$ and the smallest is $\lambda_1 \approx -\pi^2 \Delta t/(q+1)^2 \approx -0.0059\Delta t$ for $q = 40$.



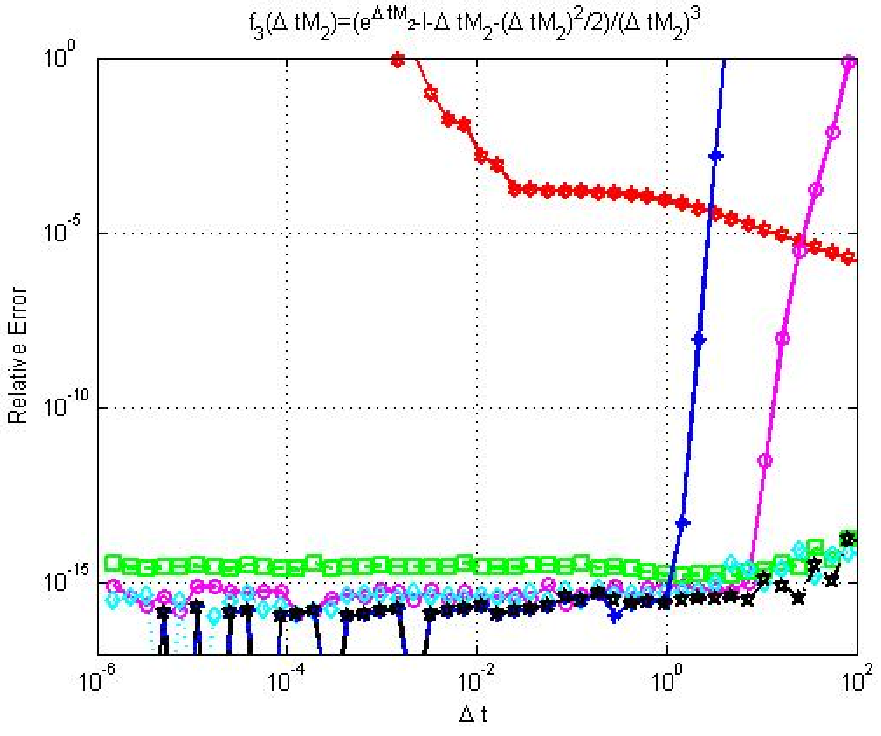


FIGURE 4.8: Relative errors in $f_2(\Delta tM_2)$ (4.71) and $f_3(\Delta tM_2)$ (4.72) versus the values of Δt in the 40×40 matrix case. The algorithms are: Explicit Formula (red stars), 30-term Taylor series (blue circles), the Cauchy Integral Formula (magenta circles), Scaling and Squaring Type **I** based on the identities (4.20) - (4.22) (black stars), Composite Matrix (cyan diamonds) and Matrix Decomposition (green squares).

Figure 4.8 also shows that, for small values of Δt, the explicit formulas $f_k(\Delta t M_2)$ (4.73), $k = 2, 3$ are inaccurate (qualitatively similar results are found for formula $f_1(\Delta t M_2)$ (4.70)) due to the cancellation errors arising from the small eigenvalues that are close to zero. For large values of Δt, the norm of the matrix $\Delta t M_2$ (4.69) gets larger, and as already noted, the computation of the matrix exponential $e^{\Delta t M_2}$, in the explicit formula, depends on the Matlab function *expm*, which is based on the Scaling and Squaring algorithm combined with Padé approximation (4.94). In this case, the algorithm also yields inaccurate results due to the increase in the number of scalings needed to approximate the matrix exponential $e^{\Delta t M_2}$. Each scaling doubles the errors due to cancellation, truncation and rounding, resulting from the matrix inversion and the repeated matrix multiplications when using the Padé approximation. The analysis of the rounding errors in using the Scaling and Squaring algorithm for approximating the exponential function (see also formula (4.30)) is explained in §4.2.3.

In the matrix case, there is a large range of values of Δt for which both the explicit formulas and the Taylor series algorithm are inaccurate, so we cannot simply switch between the two algorithms in this case as we proposed in the scalar case in §4.2.1.

4.3.2 the Cauchy Integral Formula

A less well known Cauchy integral formula is the matrix form

$$f(\Delta t M) = \frac{1}{2\pi i} \int_\Gamma \frac{f(T)}{TI - \Delta t M} dT, \tag{4.77}$$

where f is an analytic function of the matrix $\Delta t M$, I is the $q \times q$ identity matrix and the contour Γ is sufficiently large to enclose all the eigenvalues of the matrix $\Delta t M$ (see [32, 44, 45]). Formula (4.77) is an analogous to the formula (4.11), for the scalar case, presented in §4.2.2.

Suitable contours Γ may vary from one problem to another. For example, elliptical contours were investigated by **Kassam** and **Trefethen** [44, 45] and **Livermore** [53]. The ellipse is centered at some point $z_0 = x_0 + iy_0$ in the complex plane and has a semi major axis a and a semi minor axis b and can be expressed parametrically as

$$T(\theta) = z_0 + a\cos\theta + ib\sin\theta, \qquad 0 \le \theta \le 2\pi.$$

Plugging this into the Cauchy integral formula (4.77) and employing the periodic **Trapezium Rule** (4.14) to approximate the integral we obtain the formula for an elliptical contour,

$$f(\Delta t M) \approx \frac{1}{N}\sum_{j=1}^{N}(b\cos\theta_j + ia\sin\theta_j)(T(\theta_j)I - \Delta t M)^{-1} f(T(\theta_j)), \tag{4.78}$$

where $T(\theta_j) = z_0 + a\cos\theta_j + ib\sin\theta_j$, $\theta_j = 2\pi j/N$ are N points along the bounding ellipse.

The simplest choice of the contour Γ is a circle with radius R centered at some point z_0 on the real line. By making the substitution $dT(\theta) = T_\theta(\theta)d\theta$, where $T(\theta) = z_0 + Re^{i\theta} : 0 \le \theta \le 2\pi$, the Cauchy integral (4.77) becomes

$$\begin{aligned} f(\Delta t M) &= \frac{1}{2\pi i}\int_0^{2\pi} \frac{f(z_0 + Re^{i\theta})}{T(\theta)I - \Delta t M} Rie^{i\theta} d\theta, \\ &= \frac{1}{2\pi}\int_0^{2\pi} (T(\theta) - z_0)(T(\theta)I - \Delta t M)^{-1} f(T(\theta)) d\theta. \end{aligned} \tag{4.79}$$

Employing the periodic **Trapezium Rule** (4.14) to approximate the integral on the right-hand side of (4.79), we obtain the corresponding formula proposed by **Kassam** and **Trefethen** [44, 45] for a circular contour

$$f(\Delta t M) \approx \frac{1}{N}\sum_{j=1}^{N}(T(\theta_j) - z_0)(T(\theta_j)I - \Delta t M)^{-1} f(T(\theta_j)), \tag{4.80}$$

where $T(\theta_j) = z_0 + Re^{i\theta_j}$, $\theta_j = \frac{2\pi j}{N}$ are the N points around the circumference of the circle centered at z_0.

To approximate the function $f_k(\Delta t M), k = 1, 2, \ldots, s$ (required for the ETD methods of order s) with this algorithm, we simply evaluate the scalar function $f_k(z)$ (4.2), $k = 1, 2, \ldots, s$ respectively at a set of N points $T(\theta_j) = z_0 + Re^{i\theta_j}$ in the complex plane, and then apply (4.80)

$$f_k(\Delta t M) \approx \frac{1}{N}\sum_{j=1}^{N}(T(\theta_j) - z_0)(T(\theta_j)I - \Delta t M)^{-1} f_k(T(\theta_j)). \tag{4.81}$$

Our experience shows that many different choices of the contour work well, so long as one is careful to ensure that none of the points on the contour are close to or at the origin (otherwise the original problem of rounding errors reappears), and that all the eigenvalues of the matrix $\Delta t M$ are indeed enclosed by Γ.

However, formula (4.81) shows that, in order to do this, we need to work out N matrix inverses $(T(\theta_j)I - \Delta t M)^{-1}$, and this consequently restricts the good performance of the algorithm to matrices of moderate norm. This is because of the approximation of the integral (4.77) for matrices with large norm (the spread of the eigenvalues increases) via the circular contour algorithm (4.81), requires us to enlarge the circle so that it encloses all the eigenvalues of the matrices. Consequently we must increase the number N of points around the circle required to give accurate results. We therefore also increase the amount of work required for computing the large number N of matrix inverses (one for each point on the discretized circle). This adds a disadvantage in terms of the high cost in computer time (see §4.6).

In addition to the difficulties mentioned above, the eigenvalues (if not already known) must be computed beforehand – or at least, the eigenvalue of largest absolute value must be determined – in order to choose a suitable integration contour (for the matrices we consider, the eigenvalues are already known). Some of these difficulties were also noted by **Livermore** [53]. However, **Kassam** and **Trefethen** [44, 45] recommended that, if the functions that we want to calculate are real, we can halve the amount of work by exploiting the $\pm i$ symmetry of the algorithm (4.81) and evaluate in equally spaced points on the upper half of a circle centered on the real axis, then take the real part of the results. Also, **Schmelzer** and **Trefethen** [69, 70] had a new perspective on contour integrals that improves some of these difficulties. The authors have shown that the function $f_k(\Delta t M), k = 1, 2, \dots, s$ can be evaluated efficiently using a **Hankel** contour and a different form of the integral (4.77). Rather than working with circles and ellipses as contours, they enclosed the eigenvalues by open contours winding around the negative real line. The authors claimed that the use of **Hankel** contours in the Cauchy integral avoids the expensive computation of eigenvalues to estimate the shape of an enclosing contour, and overcomes the algebraic decay of the functions in the left half-plane which makes this approach flexible and efficient. Unfortunately, we received this information too late to incorporate it in our experiments.

In our experiments, we take the contour to be a circle centered at half the minimum eigenvalue (λ_{min}) of the matrix $\Delta t M_2$ (4.69) (the eigenvalues of the matrix $\Delta t M_2$ are on the negative real axis), and sampled at 128 equally spaced points in (4.81). The radius

$$R = -\frac{\lambda_{min}}{2} + 5,$$

is varying for each value of Δt to ensure that the circular contour encloses all the eigenvalues of the matrix $\Delta t M_2$ and does not pass too close to any. The above choice of R was found to be suitable for values of $\Delta t > 0.6$, but less accurate for small values of Δt. An interesting observation from our practical experiments is that the algorithm is sensitive to the choices of the center and the radius of the circular contour relative to the range of the eigenvalues of the matrix $\Delta t M_2$. For values of $\Delta t \leq 0.6$ the contour is a circle centered at the minimum eigenvalue (λ_{min}) of the matrix $\Delta t M_2$ sampled at 128 equally spaced points. The radius in this case

$$R = -\lambda_{min} + 1,$$

also varies for each value of Δt to ensure that the circular contour encloses all the eigenvalues of the matrix $\Delta t M_2$ and that the algorithm yields the desired error levels.

Regarding the test described in §4.3, we find that, when computing the numerical relative errors (4.10) of using this algorithm to approximate the expression $f_k(\Delta t M_2)$ (4.73), $k = 2, 3$ for matrix size $q = 40$ and small values of Δt, the algorithm (4.81) performs very well and the results are very satisfactory, see figure 4.8 (qualitatively similar results are found for formula $f_1(\Delta t M_2)$ (4.70)). However, this algorithm is slightly less accurate than the Scaling and Squaring algorithm type **I** and the Composite Matrix algorithm (to be described in §4.3.4 and §4.3.6 respectively), and the deficiency of its performance is particularly pronounced for large values of Δt. As is apparent in figure 4.8, there is a sharp increase of the relative errors, due to enlarging the circular contour to enclose all the eigenvalues of the matrix $\Delta t M_2$, without increasing the number N of points around the circle (more than 128 points are needed to give accurate results; in fact 512 points are needed for $\Delta t = 100$). The form of this error is analyzed in the following section.

4.3.3 Varying the Radius of the Circular Contour

To investigate the effects of varying the circular contour radius, we set up two experiments, one for the scalar case and one for the matrix. For the first, we use the Cauchy integral algorithm (4.16) to compute the scalar expression $f_k(z)$ (4.2) of orders $k = 1, 2, 3$ for a fixed number of points $N = 32$ and a fixed value of $z = 10^{-1}$ (the circle center). We start with a radius $R = 1$ and work up to a radius $R = 20$. In figure 4.9 we plot the relative errors (4.10) for each value of the radius R, where the 'exact' values of these expressions were calculated using 50 digit arithmetic. With increasing the radius R for a fixed number of discretization points N, we observe the huge growth of the errors.

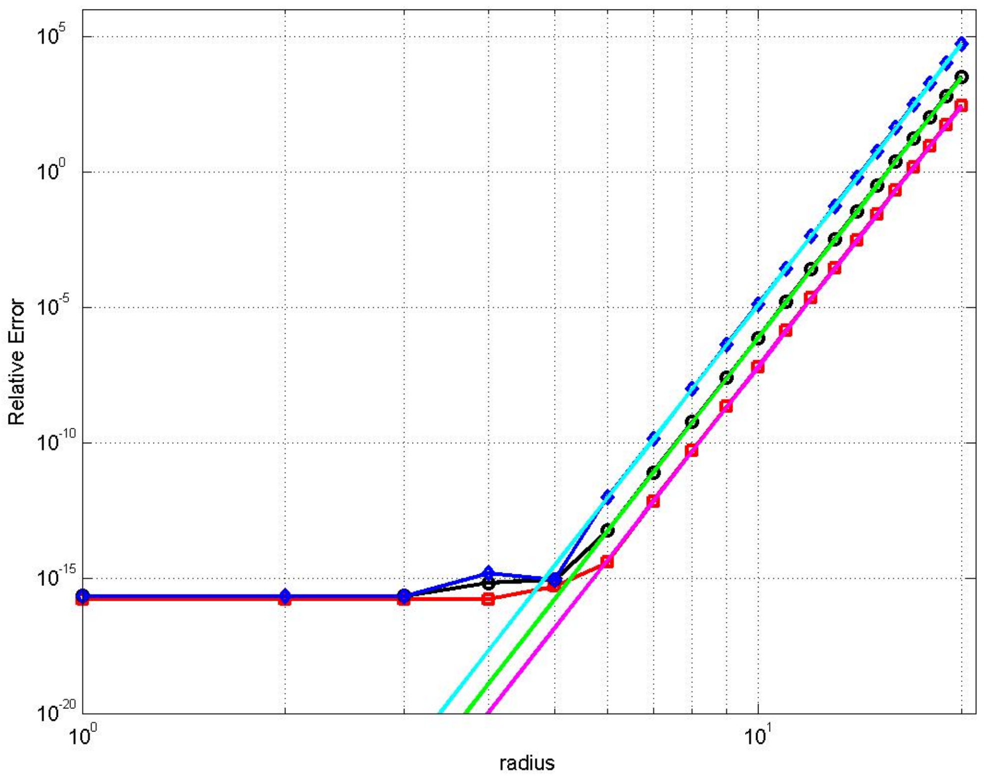


FIGURE 4.9: Relative errors of using the Cauchy integral formula (4.16), for $z = 10^{-1}$ and fixed number of points $N = 32$, versus the contour radius R, for approximating in the scalar case, the expressions: $f_1(z)$ (4.1) (blue diamonds), $f_2(z)$ (4.4) (black circles) and $f_3(z)$ (4.5) (red squares). The estimated error lines are E_1 (4.86) (cyan), E_2 (4.87) (green) and E_3 (4.88) (magenta).

For the second experiment, we use the matrix Cauchy integral formula (4.81) to compute the expression $f_k(\Delta t M_2)$ (4.73), $k = 1, 2, 3$ and $q = 40$, for a fixed number of points $N = 32$ and fixed value of $\Delta t = 0.25$, with the circle centered at zero. We again start with a radius $R = 1$ and work up to a radius $R = 20$. In figure 4.10 we plot the relative errors (4.10), using the $2-$norm (4.75) of a matrix in Matlab code for each value of the radius. As usual, the 'exact' values of these expressions were calculated using 50 digit arithmetic. The experiment shows that changing the radius R of the contour for a fixed number of discretization points N has a dramatic effect on the errors. Firstly, if we decrease the radius R so that it is too small to enclose all the eigenvalues of the matrix, we see the huge growth of the errors. Secondly, when the radius R is just enough to enclose all the matrix eigenvalues, the errors are minimized and the accuracy is good. Thirdly, with increasing the radius

R far beyond the eigenvalue with maximum absolute value, we see the errors grow unboundedly again, in the same way as in the scalar case.

We can explain this increase in the error of the algorithm with R by an examination of the leading error term in the periodic Trapezium rule (4.14)

$$\frac{1}{2\pi}\int_0^{2\pi} P(\theta)d\theta \approx \frac{1}{N}\sum_{j=1}^{N} P(\theta_j), \qquad \theta_j = \frac{2\pi j}{N}. \tag{4.82}$$

$P(\theta)$ is a periodic function of θ, so it can be written as a Fourier series

$$P(\theta) = \sum_{n=0}^{\infty} a_n e^{in\theta}.$$

Plugging this into (4.82) and interchanging the order of summation, we have

$$\frac{1}{N}\sum_{j=1}^{N}\sum_{n=0}^{\infty} a_n e^{in\theta_j} = \frac{1}{N}\sum_{n=0}^{\infty} a_n \sum_{j=1}^{N} e^{2\pi jni/N}. \tag{4.83}$$

The second summation in the last expression above is simply the sum of the N roots of unity. This is zero in general, unless the exponent n is an integer multiple K of N, i.e. $n = NK$. Therefore, the periodic Trapezium rule (4.83) gives us

$$\frac{1}{N}(Na_0 + Na_N + Na_{2N} + \cdots). \tag{4.84}$$

Equivalently, in terms of aliasing errors [84], with N points we cannot distinguish between the constant function 1 and the functions $(e^{2\pi jni/N},\ n = NK)$, since these functions are 1 at all mesh points θ_j. In addition, because of the exponential decay [79] of the Fourier coefficients, we deduce that the coefficient a_{2N} is much less than a_N. Therefore, since the true value in the periodic Trapezium rule (4.84) is a_0, the leading error term is just a_N and the relative leading error term is $|a_N/a_0|$.

We use this theory to estimate the error when using the Cauchy integral formula to approximate the scalar expression $f_1(z)$ (4.1) with a fixed number of points N, while increasing the contour radius R. We have

$$f_1(z + Re^{i\theta}) = \frac{e^z e^{Re^{i\theta}} - 1}{z + Re^{i\theta}}, \tag{4.85}$$

and if we assume that $|z| \ll R$, we can neglect z and write the right-hand side of (4.85) as a Fourier series

$$1 + Re^{i\theta}/2! + R^2e^{2i\theta}/3! + \cdots + R^N e^{Ni\theta}/(N+1)! + \cdots.$$

Hence the estimated leading relative error in the trapezium rule, $|a_N/a_0|$, is the coefficient of $e^{Ni\theta}$, which is

$$E_1 = R^N/(N+1)!. \tag{4.86}$$

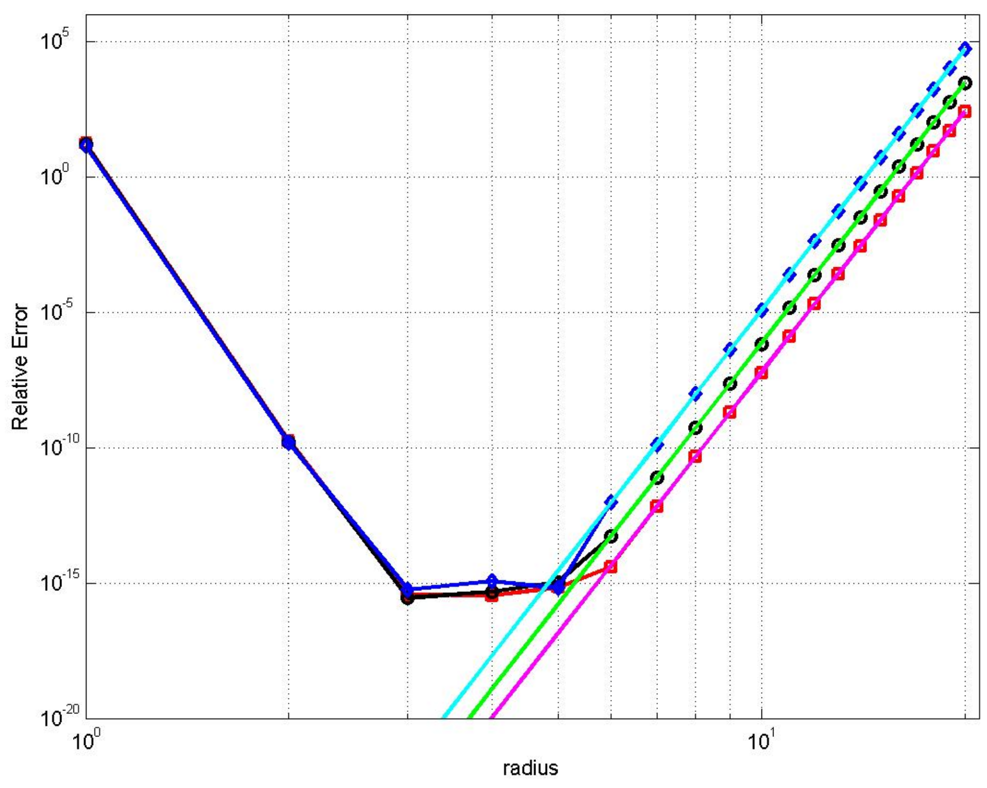


FIGURE 4.10: Relative errors of using the Cauchy integral formula (4.81), for $\Delta t = 0.25$, $q = 40$ and fixed number of points $N = 32$, versus the contour radius R, for approximating in the matrix case, the expressions: $f_1(\Delta t M_2)$ (4.70) (blue diamonds), $f_2(\Delta t M_2)$ (4.71) (black circles) and $f_3(\Delta t M_2)$ (4.72) (red squares). The estimated error lines are E_1 (4.86) (cyan), E_2 (4.87) (green) and E_3 (4.88) (magenta).

Similar calculations can be made for the expression $f_k(z)$ (4.2) of orders $k = 2, 3$, and the leading relative errors for these cases are found to be

$$E_2 = 2R^N/(N+2)!, \tag{4.87}$$

and

$$E_3 = 6R^N/(N+3)!, \tag{4.88}$$

respectively.

Figure 4.9 shows that the theoretically estimated errors E_1 (4.86), E_2 (4.87) and E_3 (4.88) agree very well with the numerical relative errors of using the Cauchy integral algorithm (4.16) for approximating the expression $f_k(z)$ (4.2) of orders $k = 1, 2, 3$ respectively, for large radius R at fixed values of discretization points N.

On the other hand, applying the same theory to estimate the error when using the Cauchy integral formula to approximate the expression $f_k(\Delta t M),\ k = 1, 2, \ldots, s$ (4.73) in the matrix case is cumbersome, though our numerical experiments show that the above analysis and results hold. Figure 4.10 shows that the theoretically estimated errors E_1, E_2 and E_3 agree very well with the numerical relative errors of using the Cauchy integral algorithm (4.81) for approximating the expressions $f_1(\Delta t M_2)$ (4.70), $f_2(\Delta t M_2)$ (4.71) and $f_3(\Delta t M_2)$(4.72) respectively.

In a third experiment, we found that two criteria need to be met for the error formulas E_1 (4.86), E_2 (4.87) and E_3 (4.88) to agree accurately with the numerical relative errors of using the Cauchy integral algorithm (4.16) for approximating the expressions[7]:

1. In the scalar case $|z| \ll 1$ and $|z| \ll R$,
2. The center of the circular contour, in the non-diagonal matrix case, should be zero.

If one of the criteria is breached, the theoretically estimated errors E_1, E_2 and E_3 will not agree with the numerical relative errors of using the Cauchy integral algorithm for approximating the expression $f_k(z)$, for large radius R at fixed values of discretization points N. Figure 4.11 shows a case of testing the Cauchy integral formula (4.81) for computing $f_1(\Delta t M_2)$ (4.70), $f_2(\Delta t M_2)$ (4.71) and $f_3(\Delta t M_2)$ (4.72) with $q = 40$, for a fixed number of points $N = 128$ and fixed value of $\Delta t = 10$. Here the circular contour is centered at half the minimum eigenvalue (λ_{min}) of the matrix $\Delta t M_2$ (4.69). In the plot, we can see that the estimated error lines E_1 (4.86), E_2 (4.87) and E_3 (4.88) do not agree with the numerical relative errors for each value of the radius, ranging from $R = -\frac{\lambda_{min}}{2} + 1$ up to $R = -\frac{\lambda_{min}}{2} + 60$.

Our error formulas can be used to determine the value of the radius R at which the algorithm becomes inaccurate for a given value of N. More usefully, for larger values of the radius R, we can also estimate the number of points N required to achieve a relative error of some chosen tolerance ϵ, in terms of the radius R and ϵ. For large integers N, we use **Stirling's** formula [1]

$$N! \approx \sqrt{2\pi N}\frac{N^N}{e^N},$$

[7]These criteria are not required for the accuracy of the Cauchy integral algorithm when approximating the expressions.

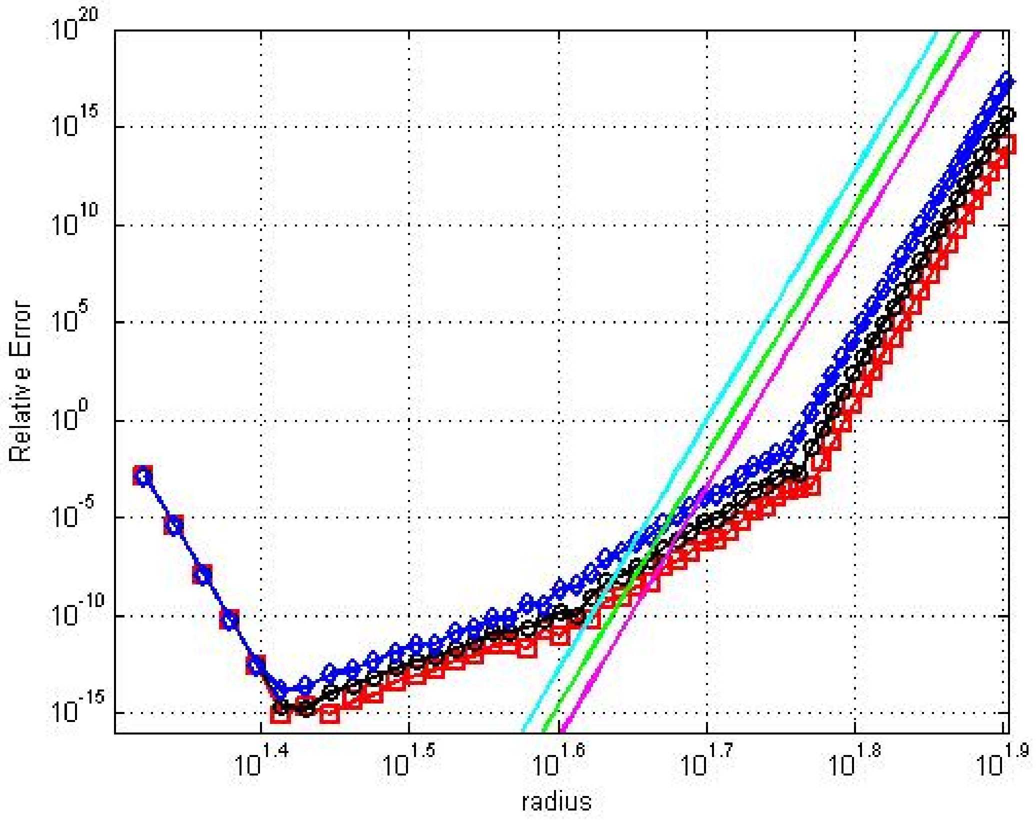


FIGURE 4.11: Relative errors of using the Cauchy integral formula (4.81), for $\Delta t = 10,\ q = 40$ and fixed number of points $N = 128$, versus the contour radius R, for approximating in the matrix case, the expressions: $f_1(\Delta t M_2)$ (4.70) (blue diamonds), $f_2(\Delta t M_2)$ (4.71) (black circles) and $f_3(\Delta t M_2)$ (4.72) (red squares). The estimated error lines are E_1 (4.86) (cyan), E_2 (4.87) (green) and E_3 (4.88) (magenta).

to approximate $(N+1)!$, in the formula $E_1 = R^N/(N+1)!$ (4.86), so that

$$\begin{aligned} R^N &\approxeq \epsilon\sqrt{2\pi}(N+1)^{N+\frac{3}{2}}e^{-(N+1)}, \\ N\log R &\approxeq \log\epsilon + \log\sqrt{2\pi} + \Big(N+\frac{3}{2}\Big)\Big[\log N + \log\Big(1+\frac{1}{N}\Big)\Big] - (N+1) \quad (4.89) \end{aligned}$$

Applying the series expansion to the logarithmic function

$$\log\Big(1+\frac{1}{N}\Big) = \sum_{j=1}^{\infty}\frac{(-1)^{j+1}}{j}\Big(\frac{1}{N}\Big)^j = \frac{1}{N} + O\Big(\frac{1}{N^2}\Big),\ \Big|\frac{1}{N}\Big| < 1,$$

and substituting in (4.89), ignoring the terms of $O(1/N)$, since they are small compared to our assumption that R and N are large gives us

$$N\log R \approxeq \log\epsilon + \log\sqrt{2\pi} + \Big(N+\frac{3}{2}\Big)\log N - N. \tag{4.90}$$

Equating the largest terms in (4.90) leads to

$$\log R \approx \log N \Rightarrow R \approx c_0 N,$$

for some constant c_0. If we substitute this result in (4.90) we obtain

$$N \log c_0 \approxeq \log \epsilon + \log \sqrt{2\pi} + \frac{3}{2}\log N - N, \tag{4.91}$$

and again equating large terms leads us to

$$\log c_0 \approx -1 \Rightarrow N \approx eR.$$

Now set

$$N \approx eR + \varepsilon, \qquad \varepsilon \ll eR, \tag{4.92}$$

so that the added ε term provides a more accurate approximation. If we again substitute in (4.90), we get

$$\begin{aligned}(eR+\varepsilon)\log R \approxeq\ & \log \epsilon + \log \sqrt{2\pi} + \Big(eR + \varepsilon + \frac{3}{2}\Big)\Big[\log eR + \log{(1 + \frac{\varepsilon}{eR})}\Big] \\ & - (eR+\varepsilon),\end{aligned}$$

and applying again the series expansion to the logarithmic function in the equation above gives

$$\begin{aligned}(eR+\varepsilon)\log R \approxeq\ & \log \epsilon + \log \sqrt{2\pi} + \Big(eR + \varepsilon + \frac{3}{2}\Big)\Big[1 + \log R + \frac{\varepsilon}{eR} + O\Big(\frac{\varepsilon}{eR}\Big)^2\Big] \\ & - (eR+\varepsilon), \\ 0 \approxeq\ & \log \epsilon + \log \sqrt{2\pi} + \varepsilon + \frac{3}{2} + \frac{3}{2}\log R, \\ \varepsilon \approxeq\ & -\log \epsilon - \log \sqrt{2\pi} - \frac{3}{2} - \frac{3}{2}\log R.\end{aligned}$$

Substituting the last result for ε in (4.92) leads to the approximate condition

$$N \approxeq eR - \log \epsilon - \log \sqrt{2\pi} - \frac{3}{2} - \frac{3}{2}\log R,$$

for the error E_1 to be of order ϵ, assuming that the radius R is large.

4.3.4 Scaling and Squaring Algorithm: Type I

In the non-diagonal matrix case, we use a 30-term Taylor series, as explained in §4.3.1, to compute the expression $f_k(\Delta t M_2)$ (4.73), $k = 1, 2, 3$ if the largest absolute eigenvalue λ_{max} of the matrix $\Delta t M_2$ (4.69) is less than some threshold value δ_1. If not, we use the following Scaling and Squaring algorithm.

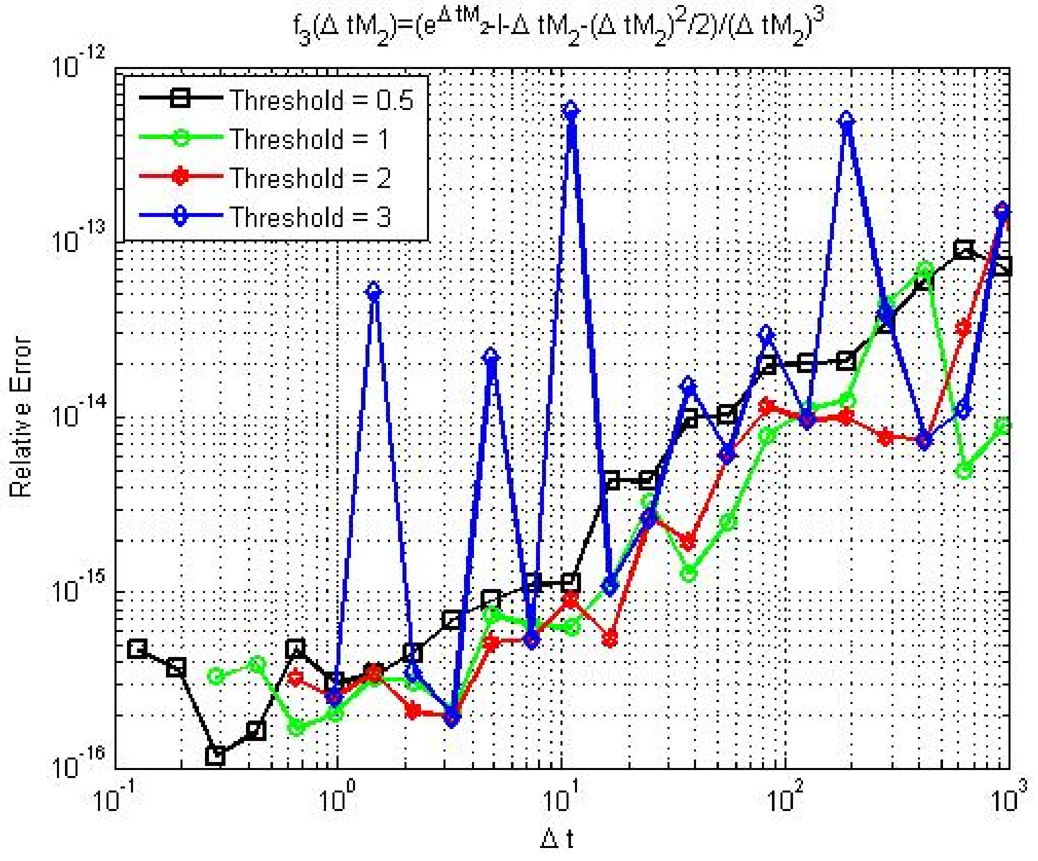


Figure 4.12: Relative errors of using the Scaling and Squaring Type **I** algorithm based on the identities (4.20) - (4.22), versus the values of Δt, for approximating the expression $f_3(\Delta t M_2)$ (4.72) for $q = 40$, for different values of threshold δ_1 (see formula (4.93)).

In a manner similar to the scalar case discussed in §4.2.3, we first use a 30-term Taylor series to compute $f_1(2^{-l_2}\Delta t M_2)$, $f_2(2^{-l_2}\Delta t M_2)$ and $f_3(2^{-l_2}\Delta t M_2)$, for some l_2 chosen to be the smallest integer such that

$$l_2 \geq \frac{\log(\lambda_{max}/\delta_1)}{\log 2}, \tag{4.93}$$

so that the largest absolute eigenvalue of the matrix $2^{-l_2}\Delta t M_2$ is less than the threshold δ_1, which we choose to be $\delta_1 = 1$ in our experiments. We then proceed by using the identities (4.20) - (4.22) or (4.23) - (4.25), or (4.27) and either the identities (4.21) - (4.22) or (4.24) - (4.25), l_2 times to compute the expression $f_k(\Delta t M_2)$ (4.73), $k = 1, 2, 3$ to obtain the final answer. Note that as for the Cauchy integral algorithm, the Scaling and Squaring algorithm requires the knowledge of the eigenvalue of largest magnitude.

Regarding the test described in §4.3, we find that the Scaling and Squaring

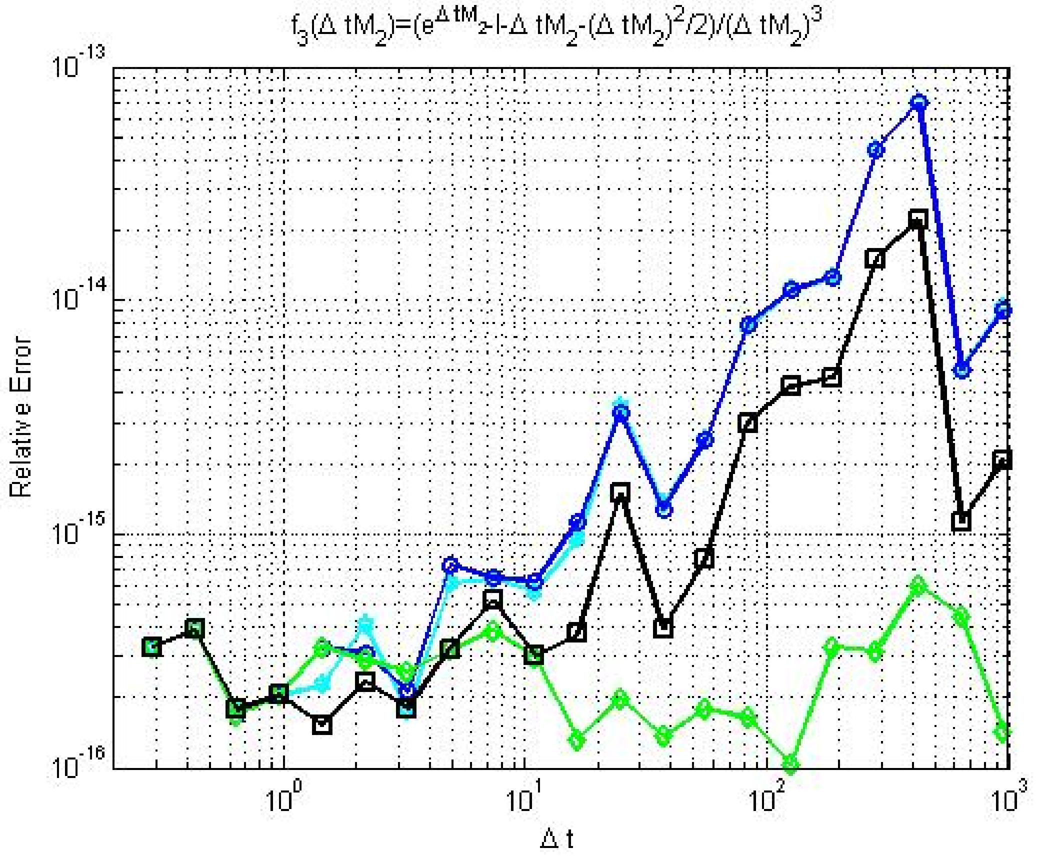


FIGURE 4.13: Relative errors of using the Scaling and Squaring Type **I** algorithm, versus the values of Δt, for approximating the expression $f_3(\Delta tM_2)$ (4.72) for $q = 40$. The blue line (circles) uses the identities (4.20) - (4.22), the cyan line (stars) uses the identities (4.23) - (4.25), the green line (diamonds) uses the identities (4.27), (4.21) and (4.22), and the black line (squares) uses the identities (4.27), (4.24) and (4.25).

algorithm based on the identities (4.20) - (4.22) is very good in the non-diagonal matrix case, being the most accurate for approximating the expression $f_k(\Delta tM_2)$ (4.73), $k = 2, 3$ for matrix size $q = 40$, for small values of Δt, as displayed in figure 4.8 (the same holds qualitatively for the expression $f_1(\Delta tM_2)$ (4.70)). The reasons for favoring this algorithm are that it is accurate and efficient for both diagonal and non-diagonal matrix problems (for small values of Δt), compared with the other algorithms. The accuracy depends on the norm of the matrix ΔtM_2 (4.69), however. As the value of Δt increases, the norm of the matrix increases. Therefore, more scaling operations are needed, leading to an amplification of the cancellation errors and the rounding errors resulting in the repeated matrix multiplication when using the Taylor expansion. In fact these errors are doubled at each scaling, and we expect the relative error to increase linearly as the value of Δt increases (in fact, the simple arguments for the scalar case in §4.2.3, about how errors do not grow for $z \ll -1$, cannot be applied directly to the matrix case).

In further tests, we compute the relative errors of using the Scaling and Squaring algorithm, based on the relations (4.20) - (4.22) or (4.23) - (4.25), or (4.27) and either the identities (4.21) - (4.22) or (4.24) - (4.25), to approximate the expression $f_k(\Delta t M_2)$ (4.73), $k = 1, 2, 3$, for different choices of the threshold values $\delta_1 = 0.5, 1, 2$. We find, firstly, that any choice of the threshold values $\delta_1 < 3$ is desirable. Figure 4.12 illustrates the relative errors of using the algorithm based on (4.20) - (4.22) for approximating $f_3(\Delta t M_2)$ (4.72) with $q = 40$, demonstrating that the accuracies of the algorithm for the threshold values $\delta_1 = 0.5, 1, 2$ are more acceptable than that for the threshold $\delta_1 = 3$. In addition, we find that there is a direct relation between larger values of the threshold and the number of terms used in the Taylor series combined with the algorithm. As the value of the threshold gets larger, it is necessary to increase the number of terms in the Taylor series to maintain the efficiency of the algorithm.

Secondly, we find that a similar level of accuracy is achieved when computing the relative errors for both families of the Scaling and Squaring formulas (4.20) - (4.22) and (4.23) - (4.25) for approximating the expression $f_k(\Delta t M_2)$ (4.73), $k = 1, 2, 3$. However, these errors are found to be larger than those resulting from using the identities (4.27), (4.21) and (4.22). These last formulas have turned out to be the most accurate out of all other formulas tested in this chapter and have the property that we need never compute a matrix exponential (the analysis, see §4.2.3, in using the Scaling and Squaring algorithm type **I**, shows that there are rounding errors (4.30) in applying the identity (4.19) to approximate a matrix exponential. However, our analysis also shows that these errors have no effect on the Scaling and Squaring algorithm type **I** based on (4.27), (4.21) and (4.22), since they do not involve computing a matrix exponential to approximate the expression $f_k(\Delta t M_2)$ for $z \ll -1$ in the scalar case. And hence, they have shown the best accuracy).

Figure 4.13 provides numerical evidence of the algorithms' validity when using the identities (4.27), (4.21) and (4.22) for approximating the expression $f_3(\Delta t M_2)$ (4.72), for matrix size $q = 40$, and a threshold value $\delta_1 = 1$, and also illustrates no significant differences between the errors for the two different forms of the scaling identities (4.20) - (4.22) and (4.23) - (4.25). However, the relative errors of using all the Scaling and Squaring formulas, except the formulas (4.27), (4.21) and (4.22), are seen to increase significantly as Δt increases[8]. This is due to the increase in the number of scalings needed (due to the increase in the norm of the matrix $\Delta t M_2$ (4.69)) to approximate the expression $f_3(\Delta t M_2)$. This process doubles (see formula (4.30)), at each scaling, the cancellation errors and the rounding errors resulting in the repeated matrix multiplication when using the Taylor expansion, and we expect the relative error to increase linearly as the value of Δt increases (the simple arguments for the scalar case in §4.2.3, about how errors do not grow for $z \ll -1$, cannot be applied directly to the matrix case). This leads us to the conclusion that, the smaller the norm of the matrix, the fewer the number of required matrix squarings, and the smaller the errors.

A similar conclusion was arrived at by **Higham** [35] who gave a new rounding error analysis that shows that the computed Padé approximant of the scaled matrix, for computing the matrix exponential, is highly accurate owing to the fact that it requires fewer matrix squarings.

4.3.5 Padé Approximation and the Taylor Series

It is more common in the literature (especially in the matrix case) to use a Padé approximation [35, 37, 47, 76] rather than Taylor series. The (n, m) Padé approximation to the exponential function $e^{\Delta t M}$ is defined by

$$r_{nm}(\Delta t M) = U_{nm}(\Delta t M)/W_{nm}(\Delta t M), \tag{4.94}$$

where $U_{nm}(\Delta t M)$ and $W_{nm}(\Delta t M)$ are polynomials of degrees at most n and m respectively, both defined as follows

$$U_{nm}(\Delta t M) = \sum_{j=0}^{n} \frac{(n+m-j)!n!}{(n+m)!j!(n-j)!}(\Delta t M)^j, \tag{4.95}$$

and

$$W_{nm}(\Delta t M) = \sum_{j=0}^{m} \frac{(n+m-j)!m!}{(n+m)!j!(m-j)!}(-\Delta t M)^j. \tag{4.96}$$

[8] We also test formulas (4.27), (4.21) and (4.22), with different values of the threshold δ_1, in approximating the function $f_k(\Delta t M)$ (4.73), $k = 1, 2, 3$, for different matrix sizes q of the Chebyshev differentiation matrix for the second derivative [11, 25, 83, 84] and of the second-order centered difference differentiation matrix $\Delta t M_1$ (4.101) for the first derivative (results are not shown). Our tests show that these formulas are the most accurate ones out of all identities used in this chapter, and that errors are seen not to increase as Δt increases. This confirms our analysis in §4.2.3.

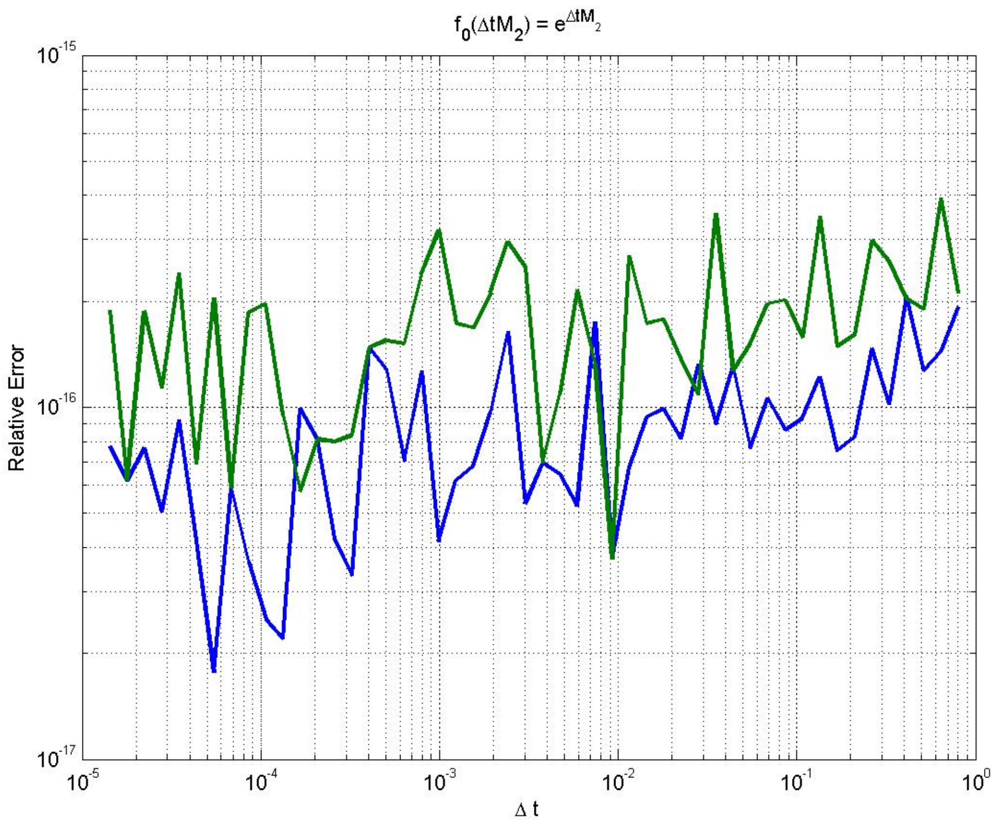


FIGURE 4.14: Relative errors of using the 16-term Taylor expansion (blue line) and the $(8, 8)$ Padé approximation (green line) versus the values of Δt, for approximating the function $f_0(\Delta tM_2) = e^{\Delta tM_2}$ for matrix size $q = 20$.

Nonsingular behavior of $W_{nm}(\Delta tM)$ (4.96) is assured if the eigenvalues of the matrix ΔtM are negative [60]. The order of the approximation is equal to the sum of the degrees of the numerator and the denominator, which matches the Taylor series expansion up to order $n + m$.

The function $f_k(\Delta tM)$ (4.73) can be approximated accurately using the Padé approximation (4.94) near the origin i.e. when the norm of the matrix ΔtM is not too large. Moreover, the diagonal Padé approximation, which uses equal degree in the numerator and the denominator is, in general, more accurate and computationally economical for a matrix argument than the off-diagonal approximation. However, we favor the Taylor series combined with the Scaling and Squaring algorithm type **I** over the Padé approximation, for three reasons. Firstly, we find that the Padé approximations lead to rounding errors roughly double those of the Taylor series, which is significant in view of the amplification of these errors caused by the scaling and squaring process, discussed in §4.2.3. For large m, $W_{mm}(\Delta tM)$ (4.96) approaches the series for $e^{-\Delta t M/2}$, whereas $U_{mm}(\Delta t M)$ (4.95) tends to the series for $e^{\Delta t M/2}$. Hence, cancellation error can reduce the accuracy. This is illustrated in figure 4.14, where we plot the relative errors of using the 16-term Taylor expansion and the (8, 8) Padé approximation to the exponential function $f_0(\Delta t M_2) = e^{\Delta t M_2}$, of the matrix M_2 (4.69) of order $q = 20$, versus the values of Δt. The exact values of the exponential function $e^{\Delta t M_2}$ are approximated using the Matlab code *expm* and 50 digit arithmetic. Secondly, in addition to the cancellation problem, the Padé approximation requires a more expensive matrix inversion. The denominator matrix $W_{nm}(\Delta t M)$ may be very poorly conditioned with respect to inversion, and this is particularly true when the matrix $\Delta t M$ has widely spread eigenvalues [60]. Thirdly, it is possible to keep the number of matrix multiplications reasonably small because of the relation $U_{nm}(\Delta t M) = W_{mn}(-\Delta t M)$, which reflects the property $1/e^{\Delta t M} = e^{-\Delta t M}$, and by using the Paterson–Stockmeyer [64] algorithm (this algorithm minimizes the number of matrix multiplications in an efficient way, by grouping the terms together and using the partitioning within a matrix polynomial; see [86] for more detail). However, we find that, when the Paterson–Stockmeyer algorithm is used, the (n, n) Padé approximation for a general function requires a number of matrix multiplications that scales as $2\sqrt{2n}$, which is exactly the same as for the corresponding Taylor series of degree $2n$.

To sum up, for the reasons mentioned above (the Padé approximation is less accurate than the Taylor series and requires a matrix inversion), we favor the Taylor series combined with the Scaling and Squaring algorithm type **I** in all of our experiments.

4.3.6 Composite Matrix Algorithm

Analogous to the scalar case (see §4.2.5), we now consider the $((s+1)q) \times ((s+1)q)$ composite matrix

$$B_s = \begin{pmatrix} \Delta t M & I & \underline{0} & \underline{0} & \underline{0} & \dots & \underline{0} \\ \underline{0} & \underline{0} & I & \underline{0} & \underline{0} & \dots & \underline{0} \\ \underline{0} & \underline{0} & \underline{0} & I & \underline{0} & \dots & \underline{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \dots & I \\ \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \dots & \underline{0} \end{pmatrix}, \tag{4.97}$$

where q is the order of the matrix ΔtM, $\underline{0}$ is the $q \times q$ zero matrix and I is the $q \times q$ identity matrix. If we exponentiate the matrix B_s, the resulting matrix

$$e^{B_s} = \begin{pmatrix} e^{\Delta tM} & f_1(\Delta tM) & f_2(\Delta tM) & f_3(\Delta tM) & f_4(\Delta tM) & \dots & f_s(\Delta tM) \\ \underline{0} & I & I & I/2 & I/3! & \cdots & I/(s-1)! \\ \underline{0} & \underline{0} & I & I & I/2 & \cdots & I/(s-2)! \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \cdots & I \end{pmatrix}, \tag{4.98}$$

returns the coefficient $f_k(\Delta tM)$ (4.73), $k = 1, 2, \dots, s$, required by the ETD methods of order s, which can again be extracted easily. The proof of the resulting matrix (4.98) is essentially the same as in the scalar case (see §4.2.5), which uses the Taylor series expansion of the exponential function. We note in particular that, due to the structure of the matrix B_s, any power of the matrix B_s contains as a sub-matrix the corresponding power of the matrix ΔtM in the same position where B_s contains ΔtM, and therefore, the exponential of the matrix ΔtM will be generated in the same position.

This algorithm is implemented using the Matlab function *expm* to approximate the exponential e^{B_s} (4.98), and has the advantage of being one of the simplest of all the algorithms to code. Figure 4.8 shows that, when computing the relative error (4.10) of using this algorithm for approximating the expression $f_k(\Delta tM_2)$ (4.73), $k = 2, 3$ of matrix size $q = 40$ for small values of Δt, the results are very satisfactory (the same holds qualitatively for the expression $f_1(\Delta tM_2)$ (4.70)). As the value of Δt increases, the norm of the matrix B_s (4.97) also increases. Therefore more scaling operations are needed[9], leading to an amplification of the cancellation errors and the rounding errors resulting from the matrix inversion and the repeated matrix multiplications when using the Padé approximation, see §4.3.5, and to an increase in the computational expense. In fact, referring to formula (4.30), these errors are doubled at each scaling and we expect the relative error to increase linearly as the value of Δt increases (see in §4.2.3 the analysis of the rounding errors in using the Scaling and Squaring algorithm for approximating the exponential function). In addition, the algorithm uses a much larger matrix than the other algorithms,

[9]As already noted, the Matlab code *expm* uses a scaling and squaring method combined with Padé approximation (4.94) [35, 37, 47, 76].

because the order of the matrix B_s (4.97) is $(s+1)$ times that of the matrix $\Delta t M_2$ (4.69). This also leads to a significant increase in the computational effort that slows the algorithm (see §4.6).

4.3.7 Matrix Decomposition Algorithm

One class of efficient algorithms for problems involving large matrices and evaluation of the exponential $e^{\Delta t M}$ is based on factorizations or decompositions [60] of the matrix $\Delta t M$.

Such matrix decompositions are based on transformations of the form

$$\Delta t M = V D V^{-1},$$

and the power series definition of $e^{\Delta t M}$ then implies

$$e^{\Delta t M} = V e^{D} V^{-1}.$$

The idea is to find a matrix V for which e^D is easy to compute. This provides a useful algorithm in the case where matrices can be diagonalized. The simplest approach [60] is to take V to be the matrix whose columns are the eigenvectors of the matrix $\Delta t M$, that is

$$V = [v_1] \ldots [v_q],$$

and

$$\Delta t M v_j = \zeta_j v_j, \qquad j = 1, \ldots, q,$$

where ζ_j are the eigenvalues of the matrix $\Delta t M$ of order q. These q equations can be written

$$\Delta t M V = V D,$$

where $D = diag(\zeta_1, \ldots, \zeta_q)$. The exponential of the diagonal matrix D can be found easily, since it only requires computing the exponential of a scalar

$$e^D = diag(e^{\zeta_1}, \ldots, e^{\zeta_q}).$$

Using the above considerations, we can write the expression $f_k(\Delta t M)$ (4.73),

$k = 1, 2, \ldots, s$ as follows:

$$\begin{aligned}
f_k(\Delta tM) &= \Big(e^{\Delta tM} - \sum_{j=0}^{k-1} \frac{(\Delta tM)^j}{j!}\Big)/(\Delta tM)^k, \\
&= (VDV^{-1})^{-k}\Big(Ve^DV^{-1} - \sum_{j=0}^{k-1} \frac{(VDV^{-1})^j}{j!}\Big), \\
&= VD^{-k}e^DV^{-1} - \sum_{j=0}^{k-1} \frac{VD^{-k}D^jV^{-1}}{j!}, \\
&= VD^{-k}\Big(e^D - \sum_{j=0}^{k-1} \frac{D^j}{j!}\Big)V^{-1}, \\
&= Vf_k(D)V^{-1}. \qquad (4.99)
\end{aligned}$$

Here, we have simplified the evaluation of a function of a non-diagonal matrix exponential to that of a diagonal matrix exponential D, whose elements are the eigenvalues $\zeta_j,\ j = 1, \ldots, q$ of the matrix ΔtM.

In our numerical experiments, firstly, we use the command $[V, D] = eig\ (\Delta tM_2)$, in Matlab code, for matrix size $q = 40$, to produce a diagonal matrix D whose elements on the main diagonal are the eigenvalues $\lambda_j,\ j = 1, 2, \ldots, q$ of the matrix, and another matrix V whose columns are the corresponding q eigenvectors. Then, we use the Taylor expansion with 30 terms, as described in §4.2.1, to approximate the exponentials $e^{\lambda_j},\ j = 1, \ldots, q$ in $f_k(D)$ (4.99), $k = 1, 2, 3$ for those eigenvalues satisfying $|\lambda_j| < 1$, and the explicit formula $f_k(z)$ (4.2) of orders $k = 1, 2, 3$ respectively for those eigenvalues satisfying $|\lambda_j| \geq 1$. Finally, we compute the matrix inverse of the matrix V, using the command *inv* in Matlab code, then apply (4.99) to approximate the expression $f_k(\Delta tM_2)$ (4.73), $k = 1, 2, 3$ and find the numerical relative errors (4.10) of using this algorithm.

According to figure 4.8, this algorithm is remarkable when we compare its accuracy with that of the explicit formula $f_k(\Delta tM_2)$ (4.73), $k = 2, 3$, over all, and with that of the Taylor series and the Cauchy integral formula for large values of Δt. However, it is less accurate than the Cauchy integral formula, the Scaling and Squaring type **I** algorithm, and the Composite Matrix algorithm for small values of Δt (qualitatively similar results are found for the formula $f_1(\Delta tM_2)$ (4.71)).

The theoretical difficulty with this algorithm obviously occurs when a matrix does not have a complete set of linearly independent eigenvectors. In this case there is no invertible matrix of eigenvectors V, and the algorithm in the conventional eigenvector approach breaks down (a more general Schur decomposition can be used in this case [56]).

4.4 Chebyshev Spectral Differentiation Matrices

In this section, we carry out some tests on Chebyshev spectral differentiation matrices [11, 25, 83, 84]. The formulas for the entries of the $(Q+1)\times(Q+1)$ Chebyshev differentiation matrix for the first derivative on the Chebyshev points $x_j = \cos(j\pi/Q)$, $j = 0, 1, \ldots, Q$, $x \in [-1, 1]$ are given in [84]. To compute the Chebyshev differentiation matrix M_c for the second derivative with Dirichlet boundary conditions, we square the Chebyshev matrix for the first derivative and then strip the first and last rows and columns to obtain a matrix M_c of order $q = Q-1$. These rows and columns have no effect, since the rows are multiplied by zero and the columns are ignored. Note that these matrices are dense and have widely-spread eigenvalues.

In order to compare the results for the Chebyshev matrix M_c with those of our earlier experiments on the finite difference matrix M_2 (4.69), we re-scale M_c so that it applies to an interval of arbitrary length $q+1 = Q$ (this ensures that its eigenvalues of small magnitude are almost identical to those of M_2). Thus we work with the matrix $M_C = 4M_c/Q^2$, for the second derivative, of order $q = Q-1 = 40$.

We again use the Matlab function *expm*, to approximate the exponential function $e^{\Delta t M_C}$, the function *inv* to find $(\Delta t M_C)^{-1}$, 50 digit arithmetic to approximate the exact values of the expression

$$f_3(\Delta t M_C) = \frac{e^{\Delta t M_C} - I - \Delta t M_C - (\Delta t M_C)^2/2}{(\Delta t M_C)^3}, \qquad (4.100)$$

and we use the 2−norm of a matrix, given by (4.75), to find the numerical relative errors (4.10) of using each algorithm to approximate the expression for a range of values of Δt. In figure 4.15 we present the results for the expression $f_3(\Delta t M_C)$ (4.100) with the errors for the use of the explicit formula; this means simply evaluating the formula $f_3(\Delta t M_C)$ using the Matlab commands *expm* and *inv* with standard double precision (16 digits) arithmetic (results for the expressions $f_1(\Delta t M_C)$ (4.70) and $f_2(\Delta t M_C)$(4.71) are found to be qualitatively similar).

The test exhibits qualitatively similar results to the case of the finite difference matrix $\Delta t M_2$ (4.69), shown in figure 4.15, except that the errors are typically larger,

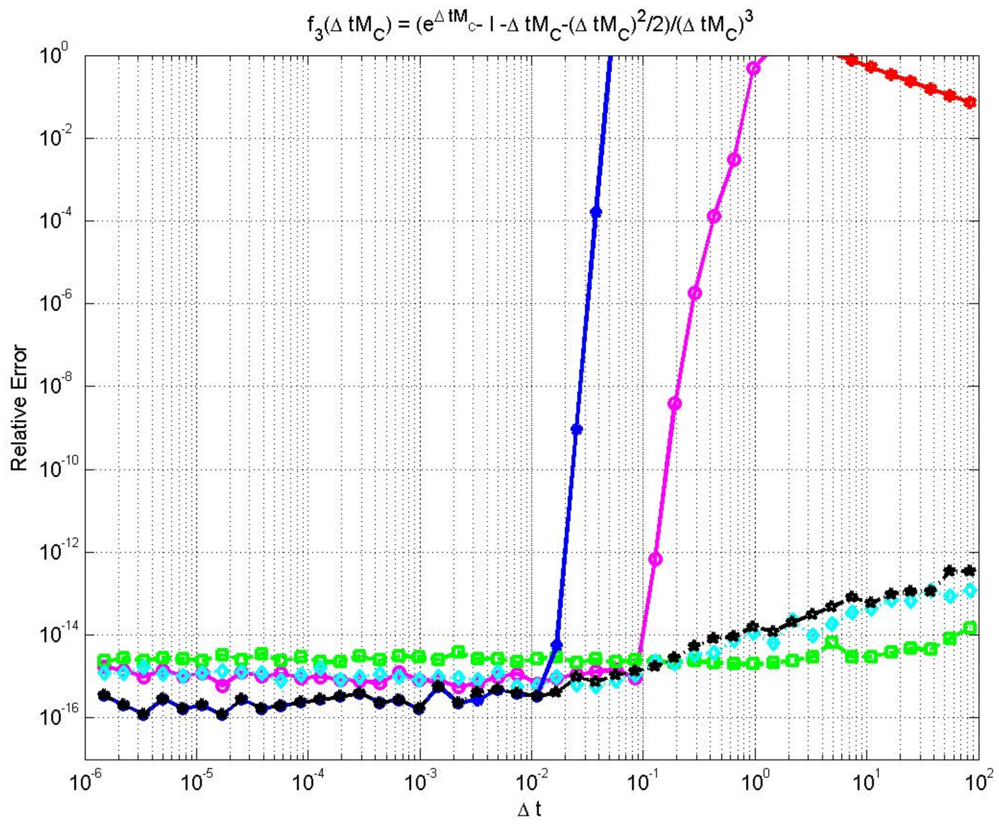


FIGURE 4.15: Relative errors for the expression $f_3(\Delta tM_C)$ (4.100) versus the values of Δt in the 40×40 matrix case. The algorithms are: Explicit Formula (red stars), 30-term Taylor series (blue circles), the Cauchy Integral Formula (magenta circles), Scaling and Squaring Type **I** based on the identities (4.20) - (4.22) (black stars), Composite Matrix (cyan diamonds) and Matrix Decomposition (green squares).

due to the larger eigenvalues of the Chebyshev matrix ΔtM_C. The eigenvalue of largest magnitude is approximately $-319.5\Delta t$ and the smallest is approximately $-0.0059\Delta t$ for $q = 40$.

The values of Δt, for which the explicit formula, the Taylor series and the Cauchy Integral Formula algorithms start to be inaccurate when numerically evaluating the function $f_3(\Delta tM_C)$, are smaller (errors are also larger and worse) than that when approximating $f_3(\Delta tM_2)$, again this is due to the larger eigenvalues of the Chebyshev matrix ΔtM_C, see figure 4.15.

For the Cauchy Integral Formula algorithm, we take the contour of integration in (4.81) to be a circle centered at half the minimum eigenvalue (ξ_{min}) of the matrix ΔtM_C (the eigenvalues of the matrix are on the negative real axis), and sampled at

128 equally spaced points. The radius

$$R = -\frac{\xi_{min}}{2} + 5,$$

is varying for each value of $\Delta t \geq 0.025$ to ensure that the circular contour encloses all eigenvalues of the matrix $\Delta t M_C$, and does not pass too close to any. The above choice of R was found to be less accurate for small values of Δt, so the radius

$$R = -\frac{\xi_{min}}{2} + 1,$$

is chosen for each value of $\Delta t < 0.025$; this again varies to ensure that the circular contour encloses all the eigenvalues of the matrix and that the algorithm yields the desired error levels.

For the Composite Matrix algorithm, we compute the exponential of the matrix B_s (4.97) (that contains the matrix $\Delta t M_C$), using the Matlab code *expm*, which is based on the Scaling and Squaring algorithm combined with Padé approximations (4.94). We find that for small values of Δt, see figure 4.15, the *expm* function leads to significantly greater rounding errors than those when using the Scaling and Squaring algorithm type **I** based on identities (4.20) - (4.22), combined with the Taylor series, with a threshold value $\delta_1 = 1$ for approximating the function $f_3(\Delta t M_C)$. This confirms our reasons, explained in §4.3.5, for favoring the Taylor series to combine the Scaling and Squaring algorithm than the Padé approximation. On the other hand, as the value of Δt increases, the norm of the matrices B_s and $\Delta t M_C$ increases, as the eigenvalues of the matrix $\Delta t M_C$ spread widely, and the performance of the Composite Matrix algorithm resembles that of the Scaling and Squaring type **I** algorithm, both being the second least accurate algorithms. This is due to the amplification (in fact, it is doubling) of the rounding errors caused by the increase in the number of scaling and squaring operations needed to approximate the function $f_3(\Delta t M_C)$ and the matrix exponential e^{B_s} (4.98). And so, we expect the relative error to increase linearly as the value of Δt increases (see in §4.2.3 the analysis of the rounding errors in using the Scaling and Squaring algorithm for approximating the exponential function, leading to formula (4.30)).

Finally, according to figure 4.15, the performance of the Matrix Decomposition algorithm surpasses that of all other algorithms for large values of Δt, though for small Δt, the algorithm's performance resembles that of the Composite Matrix algorithm, both being less accurate than the Taylor series, the Cauchy integral formula,

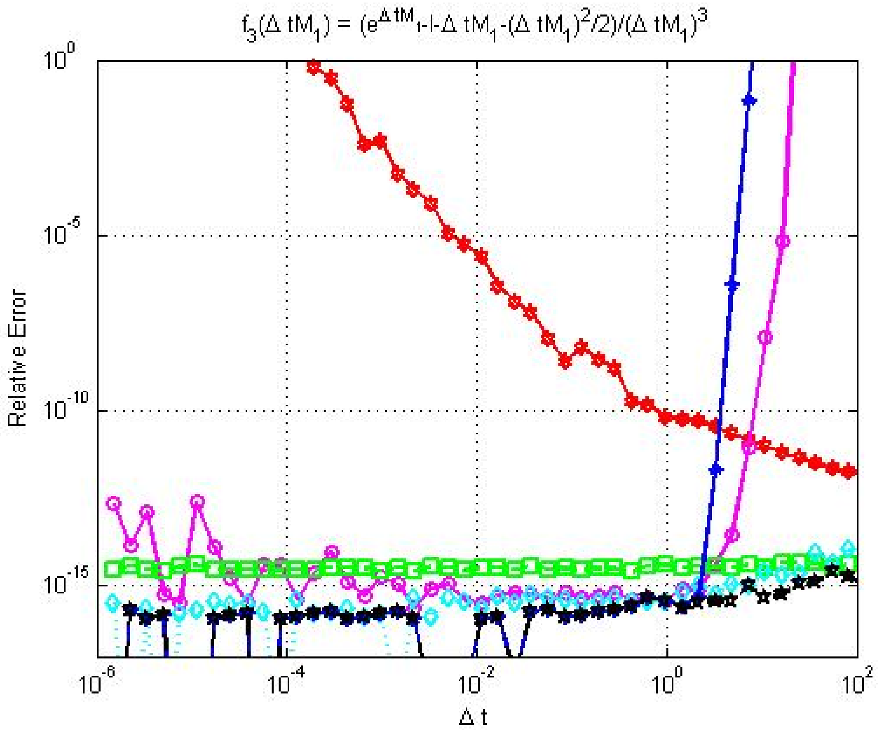


FIGURE 4.16: Relative errors in $f_3(\Delta tM_1)$ (4.102) versus the values of Δt in the 60×60 matrix case. The algorithms are: Explicit Formula (red stars), 30-term Taylor series (blue circles), the Cauchy Integral Formula (magenta circles), Scaling and Squaring Type **I** based on the identities (4.20) - (4.22) (black stars), Composite Matrix (cyan diamonds) and Matrix Decomposition (green squares).

and the Scaling and Squaring type **I** algorithm.

4.5 Matrices With Imaginary Eigenvalues

To investigate further the efficiency of the algorithms described in §4.3 for approximating the function $f_k(\Delta tM)$ (4.73), $k = 1, 2, \ldots, s$, we conduct similar tests on the 60×60 second-order centered difference differentiation matrix (see §2.2) for the first derivative,

$$M_1 = \frac{1}{2}\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & \ldots & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \ldots & -1 & 0 \end{pmatrix}. \tag{4.101}$$

Note that if the order of the matrix M_1 (4.101) is q, the scaling of the matrix M_1 is such that it corresponds to the first derivative on an interval of length $q+1$, and that the eigenvalues of the matrix are all pure imaginary. The eigenvalues η_j of the matrix $\Delta t M_1$ (4.101) can be derived analytically (see [43]) in the form

$$\eta_j = i\cos\left(\frac{j\pi}{q+1}\right)\Delta t, \qquad j = 1, \cdots, q,$$

so the eigenvalue of largest magnitude is $\eta_{max} \approx 0.998 i\Delta t$ and the smallest is $\eta_{min} \approx 0.0257 i\Delta t$ for $q = 60$.

As usual, we use the Matlab function *expm*, to approximate the exponential function $e^{\Delta t M_1}$, the function *inv* to find $(\Delta t M_1)^{-1}$, 50 digit arithmetic to approximate the exact values of the expression

$$f_3(\Delta t M_1) = \frac{e^{\Delta t M_1} - I - \Delta t M_1 - (\Delta t M_1)^2/2}{(\Delta t M_1)^3}, \tag{4.102}$$

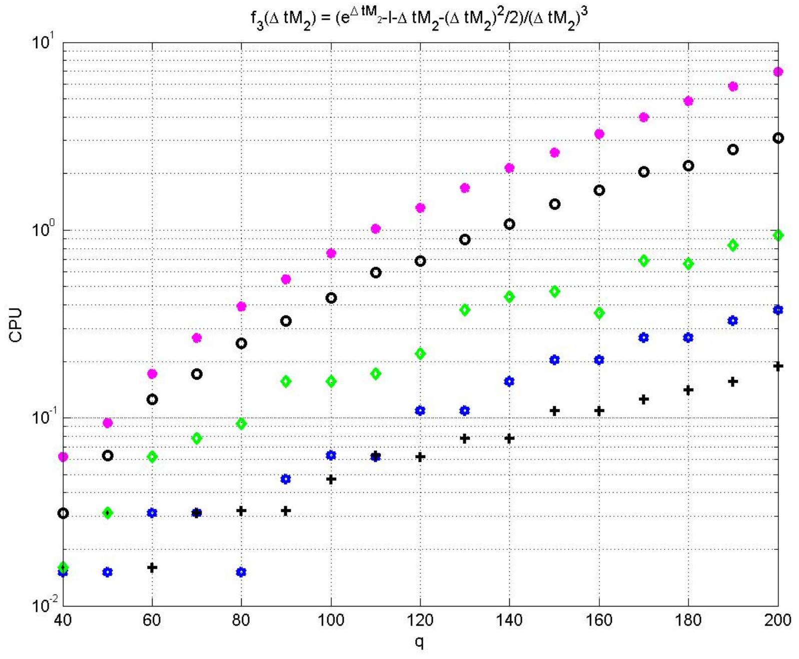
and we use the 2−norm of a matrix, given by (4.75), to find the numerical relative errors (4.10) of using each algorithm to approximate the expression. In figure 4.16 we present a comparison of the results for the expression $f_3(\Delta t M_1)$ (4.102) using the same six algorithms as in the previous sections (results for the expressions $f_1(\Delta t M_1)$ (4.70) and $f_2(\Delta t M_1)$ (4.71) are found to be qualitatively similar).

For the Cauchy Integral Formula algorithm, we take the contour of integration in (4.81) to be a circle centered at zero, and sampled at 128 equally spaced points. The radius $R = |\eta_{max}| + 3$, where $|\eta_{max}|$ is the largest absolute eigenvalue of the matrix $\Delta t M_1$ (4.101), is varying for each value of Δt to ensure that the circular contour encloses all eigenvalues of the matrix $\Delta t M_1$, and does not pass too close to any so that the algorithm yields the desired error levels.

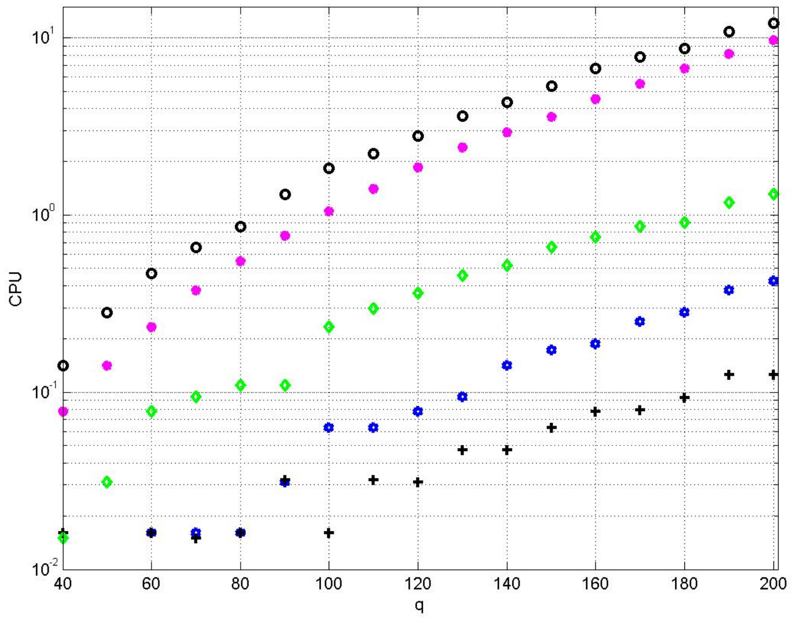
The test exhibits qualitatively similar results to the case of the finite difference matrix M_2 (4.69) and suggests that the algorithms are efficient for approximating the function $f_k(\Delta t M)$ (4.73), $k = 1, 2, \ldots, s$, for small values of Δt, whatever the type and the magnitude of the eigenvalues of the matrix M.

4.6 Computation Time

The main computational challenges in the implementation of the ETD methods are the need for fast and accurate algorithms for approximating the ETD coefficients.



(a) $\Delta t = 0.6$



(b) $\Delta t = 10$

FIGURE 4.17: CPU time of using each algorithm for approximating the expression $f_3(\Delta tM_2)$ (4.72) versus the order q of the matrix ΔtM_2 (4.69). The algorithms are: Taylor series (blue stars), the Cauchy Integral Formula (circle black), Scaling and Squaring Type **I** based on the identities (4.20) - (4.22) (green diamonds), Composite Matrix (magenta circles) and Matrix Decomposition (black pluses).

The preceding subsections investigated the accuracy of the various algorithms; we now study the computational time of each.

We calculate in Matlab code the CPU time of using each algorithm to approximate the expression (4.72)

$$f_3(\Delta t M_2) = \frac{e^{\Delta t M_2} - I - \Delta t M_2 - (\Delta t M_2)^2/2}{(\Delta t M_2)^3},$$

of the matrix $\Delta t M_2$ (4.69) versus the order of the matrix q for values ranging from $q = 40$ to $q = 200$. We perform our numerical experiments for two values of the time step Δt. In the first experiment $\Delta t = 0.6$. For the Cauchy integral formula algorithm we use $N = 32$ points to discretize the circular contour, centered at the minimum eigenvalue (λ_{min}) of the matrix $\Delta t M_2$ (the eigenvalues of the matrix M_2 are on the negative real axis) with radius $R = -\lambda_{min} + 1$ varying to enclose all the eigenvalues of the matrix. In the second experiment $\Delta t = 10$, and so the eigenvalues of the matrix $\Delta t M_2$ have a larger spread. Thus, for the Cauchy integral formula algorithm we use $N = 128$ points around the circular contour centered at half the minimum eigenvalue (λ_{min}) of the matrix $\Delta t M_2$, with radius $R = -\lambda_{min}/2 + 5$ again varying to enclose all eigenvalues.

Figure 4.17 provides the timing results for the two different values of Δt. The figure shows that most of the algorithms exhibit a CPU time proportional to q^3; this is to be expected since both matrix multiplication and matrix inversion scale in this way.

The Matrix Decomposition algorithm does extremely well over all, and is the cheapest algorithm in terms of CPU time for non-diagonal matrix problems. The majority of the time is consumed by finding the eigenvalues λ_j of the matrix $\Delta t M_2$ (4.69) and applying the 30-term Taylor series to approximate the exponentials $e^{\lambda_j},\ j = 1, \ldots, q$ in $f_3(D)$ (4.99) for those eigenvalues satisfying $|\lambda_j| < 1$.

The CPU time for the Taylor series and the Scaling and Squaring type **I** algorithms follows the same pattern for the two values of Δt. The 30-term Taylor series algorithm is the second most economical in time (having no need compute a matrix exponential), and the CPU time does not depend on the value of Δt. Most of the CPU time is spent on working out the matrix multiplications needed for the algorithm.

The Scaling and Squaring type **I** algorithm, used with a 30-term Taylor series, has a start-up time overhead, as it requires us to carry out matrix multiplications and compute several values of the identities (4.20) - (4.22) to begin. The time consumption of this algorithm depends on the norm of the matrix, the value of the threshold (here the threshold $\delta_1 = 1$), and the number of terms used in the Taylor series: when the norm gets larger (as the value of Δt increases) or when the chosen value of the threshold is smaller, more scaling operations are needed. Also, when the chosen value of the threshold is larger, more terms of the Taylor series and therefore more work on matrix multiplications are needed. Hence in both cases the algorithm becomes expensive.

The Cauchy integral formula algorithm is very expensive computationally as one has to compute N matrix inverses (128 for the largest value of Δt) and take the average (4.81) of the function values at the N points. As we enlarge the contour to enclose all the eigenvalues of the matrix, we must also increase the number of points N required to discretize the contour accurately, and therefore, the computation time of this algorithm increases.

Lastly, the Composite Matrix algorithm requires the evaluation of the exponential function e^{B_s} (4.98), which is a non-diagonal matrix of size $4q \times 4q$, so this is often the slowest algorithm. In Matlab this evaluation uses the code *expm* which depends on the Scaling and Squaring algorithm. Hence, for larger matrix order q and for larger values of Δt, i.e. larger matrix norm, more scaling operations are required, and thus the time cost increases.

4.7 Conclusion

In our investigation of the accuracy and the efficiency of six algorithms for approximating the ETD coefficients we found the following:

1. **Taylor Series:** The primary advantage of this algorithm is the simplicity and ease of implementation for both scalar and matrix cases. In addition, it is the second least costly in time. However, it is not accurate when approximating the ETD coefficients for large values (in magnitude) of the argument (matrix norm in the matrix case).

2. **The Cauchy Integral Formula:** This algorithm exhibits significant variation in performance in different cases. It has turned out to be very accurate for diagonal matrix problems, but it can be inaccurate for non-diagonal matrices

with large norm. The large errors that can arise in such case are caused by the chosen method of implementation: matrices $\Delta t M$ with large norm have a large spread of eigenvalues, and a circle of large radius is thus required to enclose all eigenvalues. This requires an a priori contour radius which in general is problem dependent, and not trivially available. In addition, the location of the eigenvalues must be known, which in general adds to the expense of the algorithm. If a fixed number of points N is used to discretize the contour then as the radius of the required circle increases, the method's accuracy decreases. To avoid this, N must be chosen to increase as the matrix norm increases (i.e. as Δt increases). This results in a large increase in computational time, as a matrix inverse has to be calculated for each point on the contour - a disadvantage to the algorithm in non-diagonal matrix cases. However, improvements to this algorithm have recently been developed [69, 70].

3. **Scaling and Squaring Algorithm Type I:** This algorithm is the most complex to implement. But it is one of the most effective and powerful algorithms for diagonal and non-diagonal matrix problems. In the non-diagonal matrix problems, knowledge of the eigenvalue of largest magnitude is required. The algorithm based on the identities (4.20) - (4.22), which has been used in our main experiments of which the results are illustrated in figures 4.8 and 4.15 - 4.17, has proved to be efficient in terms of computation time and accuracy for a good range of Δt-values, although, the errors are seen to increase in proportion to Δt. But testing the algorithm with the identities (4.27), (4.21) and (4.22) has shown that it is the most accurate out of all identities used in this chapter. Moreover, errors have not grown for a large range of Δt-values, which gives these identities an additional advantage.

4. **Scaling and Squaring Algorithm Type II:** This algorithm performs well when approximating the coefficient $f_1(z)$ that appears in the ETD1 method (3.14), but when computing the coefficients in higher order ETD methods for small values (in magnitude) of the argument the results are very inaccurate, because of the amplification of rounding errors at each scaling. Thus, the Scaling and Squaring type **II** algorithm is not a useful algorithm.

5. **Composite Matrix Algorithm:** From a practical point of view, the algorithm is successful for approximating the ETD coefficients accurately, and is

also very easy to program. However, finding the exponential of a large matrix in non-diagonal matrix problems can lead to high computational cost, caused by the larger number of operations needed to approximate the exponential matrix.

6. **Matrix Decomposition Algorithm:** For non-diagonal problems, the matrix decomposition algorithm is the cheapest algorithm in time. Most of the CPU time is spent on determining the eigenvalues required for the algorithm. Furthermore, it is remarkably accurate when compared with the explicit formula for ETD coefficients (over all Δt values considered), and with the Taylor series and the Cauchy integral formula for large values of Δt. For small values of Δt, however, it is slightly less accurate than the Cauchy integral formula, the Scaling and Squaring type **I** and the Composite Matrix algorithms.

We can sum up this set of comparisons by saying that the Scaling and Squaring type **I** algorithm is an efficient algorithm for computing the ETD coefficients in diagonal and non-diagonal matrix cases. It exhibits some loss of accuracy as the matrix norm increases, but this is much less severe than for the Taylor series and the Cauchy integral formula when approximating the ETD coefficients for large values (in magnitude) of the scalar arguments and large norm matrices respectively. Also, it compares favorably with the high computational cost of the Cauchy integral formula and the Composite Matrix algorithm in non-diagonal matrix cases. The Matrix Decomposition algorithm, in the conventional eigenvector approach, also performs well, and is very efficient computationally, though it is slightly less accurate when the matrix norm is small, and is not applicable to all matrices.

Chapter 5

Numerical Experiments

Outline of Chapter

In this chapter, we perform a variety of numerical experiments on real application problems. For the simulation tests, we choose periodic boundary conditions and apply Fourier spectral approximation for the spatial discretization. We employ first, second and fourth-order ETD methods and compare them with other competing stiff integrators including: first-order Implicit-Explicit (IMEX) method and first, second and fourth-order Integrating Factor (IF) methods for integrating in time three stiff partial differential equations (PDEs) all in one space dimension. The problems considered are: the time-dependent scalar **Kuramoto-Sivashinsky (K-S)** equation, the nonlinear **Schrödinger (NLS)** equation and the nonlinear **Thin Film** equation. In the K-S and the NLS equations, the linear terms are primarily responsible for stiffness, whereas in the third equation the nonlinear terms are the stiffest. The main testing parameters are the accuracy, the start-up overhead cost and the CPU time consumed by the methods, since these parameters play key roles in the overall efficiency of the methods.

5.1 Introduction

Over the last decade there has been a renewed interest in applying **Exponential Time Differencing (ETD)** schemes [15, 39, 53, 61, 71] to the solution of stiff systems. A Matlab package recently designed by **Berland et al.** [8] aimed to facilitate easy testing and comparison of various exponential integrators, of Runge-Kutta, multi-step and general linear type methods, applied to semi-linear problems such as, the **Kuramoto-Sivashinsky (K-S)** [41] and the nonlinear **Schrödinger (NLS)** [77] equations.

One of the main reasons for this renewed interest is the improvement in the accurate computation of the coefficients that arise in ETD schemes [2, 5, 8, 35, 47, 54, 56, 57, 67, 70, 80, 81] (this includes the exponential and related functions; see also §4). Following these efforts, the exponential integrators have emerged as viable alternatives to classical ones. The numerical comparisons presented in [81], for solving chemical kinetics problems, and the numerical experiments performed in [37], for solving large stiff systems of DEs, reveal examples where explicit exponential integrators outperform standard integrators. A similar conclusion was reached by **Du** and **Zhu** [22, 23] when they performed some simulations of micro-structure evolution (a core component of phase field modeling) in two and three dimensions. The authors found that the higher order ETD based schemes can be several orders of magnitude faster than low-order **Implicit-Explicit (IMEX)** [87] methods.

The superior performance of the ETD methods, for solving some dissipative and dispersive PDEs, was also illustrated in [19] by **Cox** and **Matthews**. In addition, **Kassam** and **Trefethen** [44, 45] compared the ETD methods of [19] with various fourth-order methods for solving various one-dimensional diffusion-type problems. They concluded that exponential integrators are highly competitive and accurate, with the best, by a clear margin, being the ETD4RK method of [19]. However, more recently **Krogstad** [49] presented an alternative fourth-order ETD method (ETDRK4-B), and found that it is slightly more accurate than the ETD4RK method of [19] when solving several semi-discretized PDEs, such as the Kuramoto-Sivashinsky (K-S) equation.

A recent report [57] on six different types of exponential integrators showed that, especially for parabolic semi-linear problems, such as the K-S and the nonlinear Schrödinger (NLS) equations, the ETD type of exponential integrators outperform integrators of Lawson type [52]. Again, **Berland** and **Skaflestad** [7] used the NLS equation as a numerical test problem, and found that under certain circumstances the performance of a fourth-order Lawson integrating factor method was demonstrably poorer than the fourth-order ETD4RK method of [19]. Further studies on solving numerically the NLS equation were presented in [46]. The author compared the performances of several fourth-order methods (mainly related to exponential integrators), and found that in specific cases, these methods can be efficiently used to solve accurately the test equations numerically.

The aim of this chapter is to make some observations regarding the efficiency of a variety of exponential integrators of different orders (including the ETD and the ETD-RK methods proposed by **Cox** and **Matthews** [19], see §3.2) when compared with other competing stiff integrators. These methods are listed in §5.2 with details of the implementations. We conduct numerical studies and comparison experiments on three model problems all in one space dimension. In §5.3 and §5.4, we consider the numerical solution of the time dependent scalar **Kuramoto-Sivashinsky (K-S)** equation [41] and the nonlinear **Schrödinger (NLS)** equation [77] respectively. The third model considered is the nonlinear **Thin Film** equation [36] (to our knowledge, no work containing the application of the exponential integrators to the thin film equation has been done). In the K-S and the NLS equation equations, the linear terms of the equations are primarily responsible for stiffness whereas in the thin film equation the nonlinear terms are the stiffest. However, we show in §5.5, that this equation can be treated within the same framework as the K-S and the NLS equations.

5.2 Numerical Experiments

Our comparison experiments are based on the simulation of three model problems, all in one space dimension: the time-dependent scalar dissipative **Kuramoto-Sivashinsky (K-S)** equation [41], the nonlinear dispersive **Schrödinger (NLS)** equation [77] and the nonlinear **Thin Film** equation [36] which is characterized as a dissipative and a dispersive PDE. All the calculations presented in this chapter are performed using Matlab codes.

For the simulation tests, we choose periodic boundary conditions. This leads conveniently to the application of the Fourier spectral approximation [11, 12, 25, 83, 84]. This approximation provides very high accuracy for smooth solutions of the test model problems. In the resulting system of ordinary differential equations (ODEs), the linear part of the model becomes diagonal, i.e. an uncoupled system of equations for each Fourier mode. The nonlinear term is transformed to physical space and evaluated at the uniform grid points and then transformed back to spectral space. Hereafter, we advance the system of ODEs in time by a numerical integration that can be used effectively in combination with the spectral approximation.

For the time discretization, all comparable methods, listed below, are expressed with respect to the model problem

$$\frac{du(t)}{dt} = cu(t) + F(u(t), t), \tag{5.1}$$

where the constant c is either large, negative and real, or large and imaginary, or complex with large, negative real part, and $F(u(t), t)$ is the nonlinear forcing term, see §3.2.

In our study of first-order accurate methods, we analyze the performance of the first-order **ETD1** method [9, 15, 19, 61]

$$u_{n+1} = u_n e^{c\Delta t} + (e^{c\Delta t} - 1)F_n/c, \tag{5.2}$$

where Δt represents the time step and u_n and F_n denote the numerical approximation to $u(t_n)$ and $F(u(t_n), t_n)$ respectively, and compare its accuracy with the **Euler** method

$$u_{n+1} = u_n + \Delta t(cu_n + F_n). \tag{5.3}$$

The Euler method is used only to obtain the numerical solution for the K-S equation [41] and the nonlinear thin film equation [36]. The comparison also includes the first-order **Integrating Factor Euler (IFEULER)** method [11, 84]

$$u_{n+1} = (u_n + \Delta t F_n)e^{c\Delta t}, \tag{5.4}$$

and the first-order **Implicit-Explicit (IMEX)** method [4] (see §1)

$$u_{n+1} = u_n + \Delta t(cu_{n+1} + F_n). \tag{5.5}$$

For second-order accurate comparison, we compare the two-step **ETD2** method [19]

$$u_{n+1} = u_n e^{c\Delta t} + \{((c\Delta t+1)e^{c\Delta t} - 2c\Delta t - 1)F_n + (-e^{c\Delta t} + c\Delta t + 1)F_{n-1}\}/(c^2\Delta t), \tag{5.6}$$

the **ETD2RK1** method [19]

$$\begin{aligned} a_n &= u_n e^{c\Delta t} + (e^{c\Delta t} - 1)F_n/c, \\ u_{n+1} &= a_n + (e^{c\Delta t} - c\Delta t - 1)(F(a_n, t_n + \Delta t) - F_n)/(c^2\Delta t), \end{aligned} \tag{5.7}$$

and the **ETD2RK2** method (derived in §3.2)

$$\begin{aligned} a_n &= u_n e^{c\Delta t/2} + (e^{c\Delta t/2} - 1)F_n/c, \\ u_{n+1} &= u_n e^{c\Delta t} + \{((c\Delta t - 2)e^{c\Delta t} + c\Delta t + 2)F_n \\ &\quad +2(e^{c\Delta t} - c\Delta t - 1)F(a_n, t_n + \Delta t/2)\}/(c^2\Delta t), \end{aligned} \tag{5.8}$$

against the second-order **ETD2CP** method introduced by **Calvo & Palencia** [15]

$$u_{n+1} = u_{n-1}e^{2c\Delta t} + \{(e^{2c\Delta t} - 2c\Delta t - 1)F_n + ((c\Delta t - 1)e^{2c\Delta t} + c\Delta t + 1)F_{n-1}\}/(c^2\Delta t), \tag{5.9}$$

and the **ETDC2** method

$$u_{n+1} = u_n e^{c\Delta t} + (e^{c\Delta t} - 1)(3F_n - F_{n-1})/2c, \tag{5.10}$$

presented by **Livermore** [53] in the solution of the incompressible magnetohydrodynamics equations. In addition, we apply the second-order **Integrating Factor Runge-Kutta (IFRK2)** method [19]

$$\begin{aligned} a_n &= \Delta t F_n e^{c\Delta t}, \\ b_n &= \Delta t F((u_n + \Delta t F_n)e^{c\Delta t}, t_n + \Delta t), \\ u_{n+1} &= u_n e^{c\Delta t} + \tfrac{1}{2}(a_n + b_n). \end{aligned} \tag{5.11}$$

For higher order ETD methods, we consider particularly the comparison of the fourth-order **ETD4** method (derived in §3.2)

$$u_{n+1} = u_n e^{c\Delta t} + (\Phi_1 F_n - \Phi_2 F_{n-1} + \Phi_3 F_{n-2} - \Phi_4 F_{n-3})/(6c^4\Delta t^3), \tag{5.12}$$

where

$$\begin{aligned} \Phi_1 &= (6c^3\Delta t^3 + 11c^2\Delta t^2 + 12c\Delta t + 6)e^{c\Delta t} - 24c^3\Delta t^3 - 26c^2\Delta t^2 - 18c\Delta t - 6, \\ \Phi_2 &= (18c^2\Delta t^2 + 30c\Delta t + 18)e^{c\Delta t} - 36c^3\Delta t^3 - 57c^2\Delta t^2 - 48c\Delta t - 18, \\ \Phi_3 &= (6c^2\Delta t^2 + 24c\Delta t + 18)e^{c\Delta t} - 24c^3\Delta t^3 - 42c^2\Delta t^2 - 42c\Delta t - 18, \\ \Phi_4 &= (2c^2\Delta t^2 + 6c\Delta t + 6)e^{c\Delta t} - 6c^3\Delta t^3 - 11c^2\Delta t^2 - 12c\Delta t - 6, \end{aligned}$$

against the **ETD4RK** method [19]

$$\begin{aligned} a_n &= u_n e^{c\Delta t/2} + (e^{c\Delta t/2} - 1)F_n/c, \\ b_n &= u_n e^{c\Delta t/2} + (e^{c\Delta t/2} - 1)F(a_n, t_n + \Delta t/2)/c, \\ c_n &= a_n e^{c\Delta t/2} + (e^{c\Delta t/2} - 1)(2F(b_n, t_n + \Delta t/2) - F_n)/c, \\ u_{n+1} &= u_n e^{c\Delta t} + \{((c^2\Delta t^2 - 3c\Delta t + 4)e^{c\Delta t} - c\Delta t - 4)F_n \\ &\quad +2((c\Delta t - 2)e^{c\Delta t} + c\Delta t + 2)(F(a_n, t_n + \Delta t/2) + F(b_n, t_n + \Delta t/2)) \\ &\quad +((-c\Delta t + 4)e^{c\Delta t} - c^2\Delta t^2 - 3c\Delta t - 4)F(c_n, t_n + \Delta t)\}/(c^3\Delta t^2), \end{aligned} \tag{5.13}$$

and the **Integrating Factor Runge-Kutta (IFRK4)** method [7, 44, 45]

$$\begin{aligned}
a_n &= \Delta t F_n, \\
b_n &= \Delta t F((u_n + a_n/2)e^{c\Delta t/2}, t_n + \Delta t/2), \\
c_n &= \Delta t F(u_n e^{c\Delta t/2} + b_n/2, t_n + \Delta t/2), \\
d_n &= \Delta t F(u_n e^{c\Delta t} + c_n e^{c\Delta t/2}, t_n + \Delta t/2), \\
u_{n+1} &= u_n e^{c\Delta t} + \tfrac{1}{6}(a_n e^{c\Delta t} + 2(b_n + c_n)e^{c\Delta t/2} + d_n).
\end{aligned} \tag{5.14}$$

Our tests are broken into two parts, each designed to show a particular aspect of the methods' performance. The goal of the first set of tests is to address the question of stability and accuracy of the methods. Therefore, we perform a series of runs with different choices of final times t which are computed, for all methods, with various time-step sizes. The time-step values are selected to ensure that all methods achieve stable accurate results. In the second set of tests, we turn our attention to the accuracy as a function of CPU time to top up the differentiation factors between the methods for each model tested. The CPU time is one of the factors that affects the efficiency of the methods, that is because a method could be stable and achieve a good accuracy in few steps, but, it could be more costly, due to the larger number of operations per time step, and consequently less efficient than others.

We measure the accuracy in terms of the relative error evaluated in the maximum norm, the 2-norm and the integrated error norm, between the results of each time stepping method (for different time-step sizes) and an "exact" solution. The relative error of the maximum norm is given by

$$relative\ max\ error = \frac{\max|numerical\ solution| - \max|exact\ solution|}{\max|exact\ solution|}, \tag{5.15}$$

the relative error of the 2-norm is given by

$$relative\ norm\ error = \frac{\left(\sum|numerical\ solution|^2\right)^{1/2} - \left(\sum|exact\ solution|^2\right)^{1/2}}{\left(\sum|exact\ solution|^2\right)^{1/2}}, \tag{5.16}$$

and the relative error of the integrated error norm is given by

$$relative\ integrated\ error = \frac{\left(\sum|numerical\ solution - exact\ solution|^2\right)^{1/2}}{\left(\sum|exact\ solution|^2\right)^{1/2}}, \tag{5.17}$$

where the sum is taken over the number of grid points in the spatial discretization. For the K-S and the nonlinear thin film equations, no explicit general analytic solutions exist, and the exact solution is approximated numerically using a fourth-order method with a very small time-step size. On the other hand, when considering the NLS equation, we focus primarily on the traveling solitons [10] as explicit exact solutions when evaluating the errors.

Experimentally, we find that the relative errors (5.15) and (5.16) do not represent appropriate measurements of accuracy. For example, the relative error (5.15) focuses on an error occurring in the difference between a local maximum point of the numerical and the exact solution. This could be misleading as it could yield a zero error even when the numerical and exact solutions are different. Therefore, the desirable choice for a measure of accuracy is the relative error of the integrated error norm (5.17). This error is more meaningful and gives a representative measure of the error in the entire solution space. Also, it does not yield a zero error unless the numerical and the exact solution agree at all points. Hence, in our tests, we plot the numerical relative error of the integrated error norm (5.17) as a function of the time step and of the CPU time for each model tested for various initial conditions.

Considering the implementation of the above time discretization methods, we find that the task is straightforward for the first-order methods. However, as the order of the methods increases, the complexity of the implementation grows. The higher order methods require more memory space, and need a relatively large computational effort. For example, the multi-step ETD and the ETD-RK methods require an accurate algorithm for evaluating the coefficients of $F(u(t_n), t_n)$ to avoid numerical difficulties (see §4). We use the 'Cauchy integral' approach (fully detailed in §4.2.2) proposed by **Kassam** and **Trefethen** [44, 45]. In this approach, we evaluate the coefficients (one coefficient for the ETDC2 method (5.10), three coefficients for the ETD2 (5.6), the ETD2CP (5.9), the ETD2RK1 (5.7) and the ETD2RK2 (5.8) methods, four coefficients for the ETD4RK (5.13) method and eight coefficients for the ETD4 method (5.12)), once at the beginning of the integration for each value of the time-step sizes, by means of contour integration in the complex plane approximated by the Trapezium rule (4.16). In addition, the multi-step ETD methods need to store the nonlinear terms at several previous time steps in order to advance the solution. So, preceding the integration loop and for each value of the time-step sizes, we obtain, for the two-step methods (ETD2, ETD2CP and ETDC2), one starting value of the nonlinear term using the ETD1 method (5.2), and three starting values of the nonlinear term for the ETD4 method using the ETD4RK method. Additionally, we store one value of the solution at the previous time step for the ETD2CP method. Moreover, the ETD2RK1, the ETD2RK2, the IFRK2 (5.11), the ETD4RK and the IFRK4 (5.14) methods carry out two (for the second-order methods) and four (for the fourth-order methods) function transforms per time step in the main loop of integration.

For the IF schemes, we find that they require the evaluation of one or more matrix exponentials, for which acceptable algorithms are well known [9, 60]. However, the schemes have some disadvantages. For example, they do not preserve fixed points for the differential equations, and are also known for having rather larger error constants [7, 11, 19, 49, 57] (for PDEs with slowly varying nonlinear terms) than other methods of the same order.

The investigation of the methods' performances and the results of the experiments are outlined in §5.3, §5.4 and §5.5 for the numerical solution of the K-S equation, the NLS equation and the nonlinear thin film equation respectively.

5.3 Kuramoto-Sivashinsky (K-S) Equation

The Kuramoto-Sivashinsky equation, which we will refer to as the K-S equation, is one of the simplest PDEs capable of describing complex (chaotic) behavior in both time and space. This equation has been of mathematical interest [29, 75] because of its rich dynamical properties. In physical terms, this equation describes reaction diffusion problems, and the dynamics of viscous-fluid films flowing along walls, and was introduced by **Sivashinsky** [74] as a model of laminar flame-front instabilities and by **Kuramoto** [50] as a model of phase turbulence in chemical oscillations. A fairly large number of numerical and theoretical studies have been devoted to the K-S equation; the reader is referred to the review paper of **Hyman** & **Nicolaenko** [41].

The K-S equation in one space dimension can be written in "derivative" form

$$\frac{\partial w(x,t)}{\partial t} = -w(x,t)\frac{\partial w(x,t)}{\partial x} - \frac{\partial^2 w(x,t)}{\partial x^2} - \frac{\partial^4 w(x,t)}{\partial x^4}, \tag{5.18}$$

or in "integral" form

$$\frac{\partial u(x,t)}{\partial t} = -\frac{1}{2}\Big(\frac{\partial u(x,t)}{\partial x}\Big)^2 - \frac{\partial^2 u(x,t)}{\partial x^2} - \frac{\partial^4 u(x,t)}{\partial x^4}, \tag{5.19}$$

where $w(x,t) = \partial u(x,t)/\partial x$.

Equation (5.18) has strong dissipative dynamics, which arise from the fourth-order dissipation ($\partial^4 w/\partial x^4$) term that provides damping at small scales. Also, it includes the mechanisms of a linear negative diffusion ($\partial^2 w/\partial x^2$) term, which is responsible for an instability of modes with large wavelength, i.e. small wave-numbers. The nonlinear advection/steepening ($w\partial w/\partial x$) term in the equation transforms energy between large and small scales.

The zero solution of the K-S equation is linearly unstable (the growth rate $\lambda(k) > 0$, for perturbations of the form $e^{\lambda t}e^{ikx}$) to modes with wave-numbers $|k| = |2\pi/\ell| < 1$ for a wavelength ℓ, and is damped for modes with $|k| > 1$, see figure 5.1; these modes are coupled to each other through the non-linear term.

We can write the K-S equation (5.18) with $2L$ periodic boundary conditions in Fourier space as follows

$$\frac{d\hat{w}_k(t)}{dt} = (k^2 - k^4)\hat{w}_k(t) - \frac{ik}{2}\mathbf{fft}(w(t)^2), \tag{5.20}$$

where **fft** is a Matlab command that represent the fast Fourier transform FFT. The stiffness in the system (5.20) is due to the fact that the diagonal linear operator, with the elements $k^2 - k^4$, has some large negative real eigenvalues that represent decay, because of the strong dissipation, on a time scale much shorter than that typical of the nonlinear term. Thus the dynamics are dominated by a relatively few large scale modes. However, we expect all methods except the Euler method to work reasonably well regarding the stability analysis.

The nature of the solutions to the K-S equation varies with the system size L. For large L, enough unstable Fourier modes exist to make the system chaotic. For small L, insufficient Fourier modes exist, causing the system to approach a steady state solution. In this case, the ETD methods integrate the system very much more accurately than the IF methods, since the ETD methods assume in their derivation that the solution varies slowly in time.

For the simulation tests, we choose two periodic initial conditions

$$w_1(x,0) = \exp(\cos(x/2)),\ x \in [0,4\pi], \tag{5.21}$$

$$w_2(x,0) = 1.7\cos(x/2) + 0.1\sin(x/2) + 0.6\cos(x) + 2.4\sin(x),\ x \in [0,4\pi]. \tag{5.22}$$

When evaluating the coefficients of the ETD and the ETD-RK methods via the 'Cauchy integral' approach [44, 45] (see §4.2.2), we choose circular contours of radius

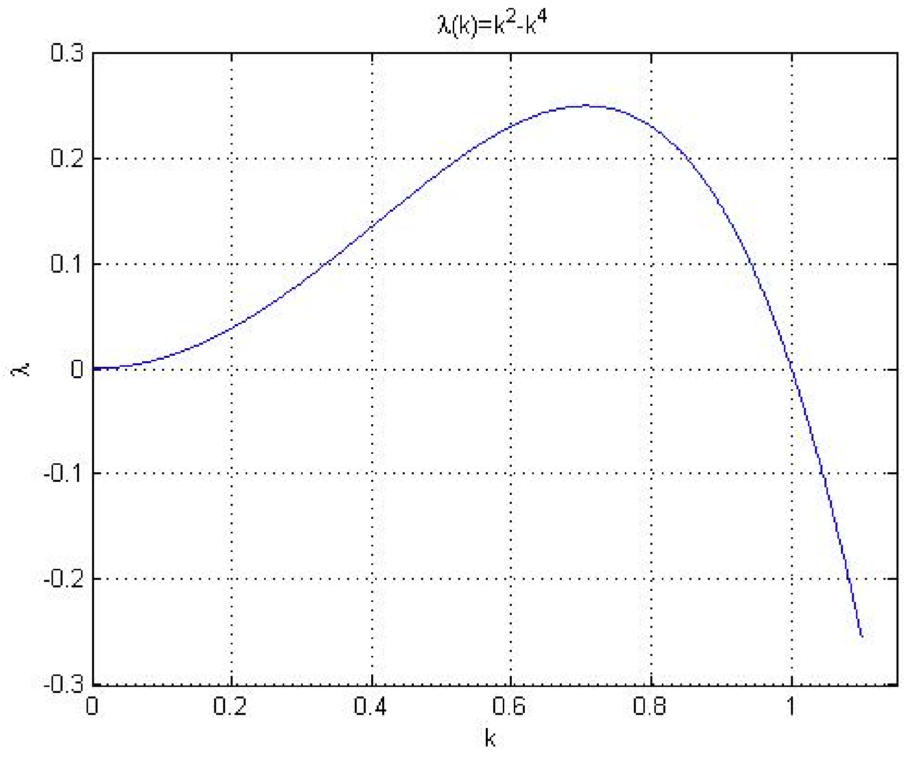


FIGURE 5.1: The growth rate $\lambda(k)$ for perturbations of the form $e^{\lambda t}e^{ikx}$ to the zero solution of the Kuramoto-Sivashinsky (K-S) equation (5.18).

$R = 1$. Each contour is centered at one of the elements that are on the diagonal matrix of the linear part of the semi-discretized model (5.20). The contours are sampled at 32 equally spaced points and approximated by (4.16).

In figure 5.2, we show the numerical solution of the K-S equation (5.18) with the initial condition $w_1(x, 0) = \exp(\cos(x/2))$, $x \in [0, 4\pi]$ (5.21), using $N_{\mathcal{F}} = 64$ grid points in the Fourier spatial discretization. We integrate the system (5.20) using the ETD4RK method (5.13) with time-step size $\Delta t = 2^{-10}$ and up to final time $t = 60$. The solution, in the figure, appears as a mesh plot and shows waves propagating, traveling periodically in time and persisting without change of shape. The computations are performed using Matlab code in a program described in Appendix A.

In the following section, we present the results of integrating the system (5.20) for the two initial conditions (5.21) and (5.22), up to final time $t = 30$, utilizing the methods described in §5.2. Again, we use $N_{\mathcal{F}} = 64$ grid points in the Fourier spatial discretization. The results are supported by figures and analysis of the methods' efficiency.

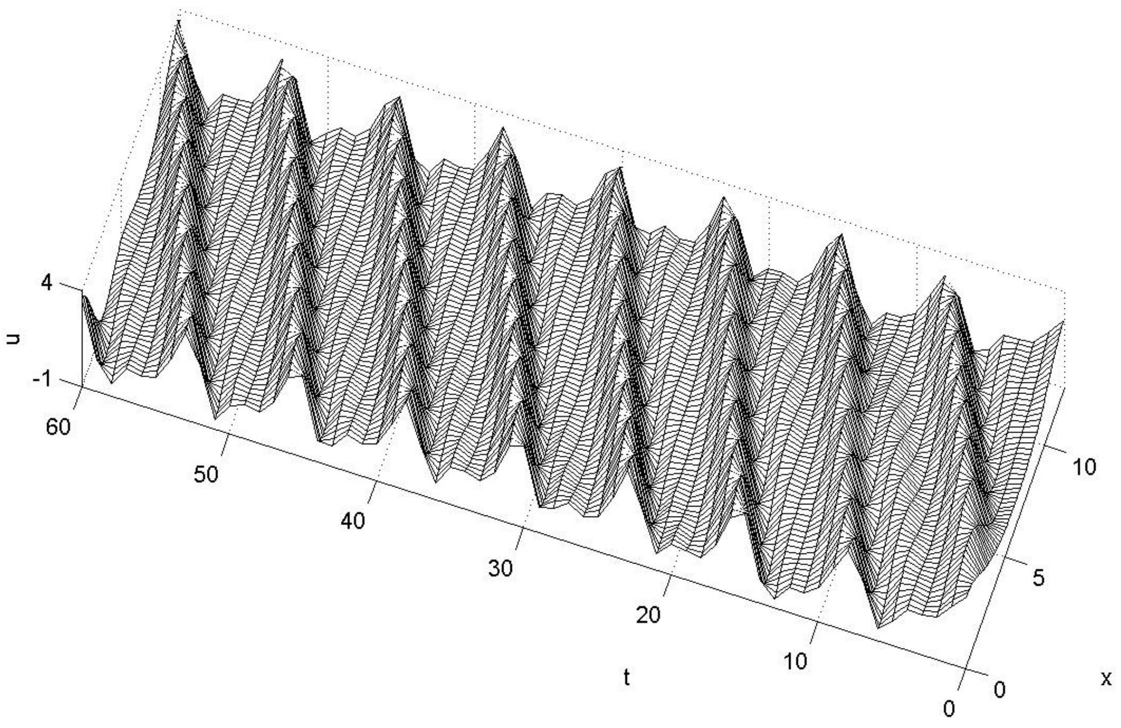
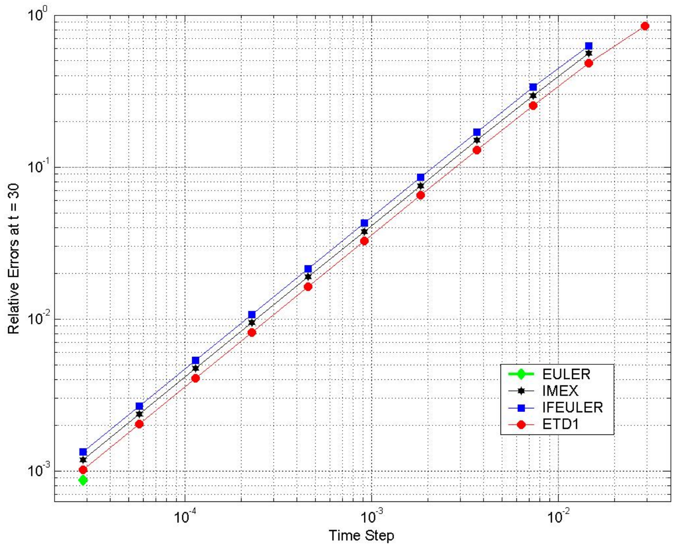
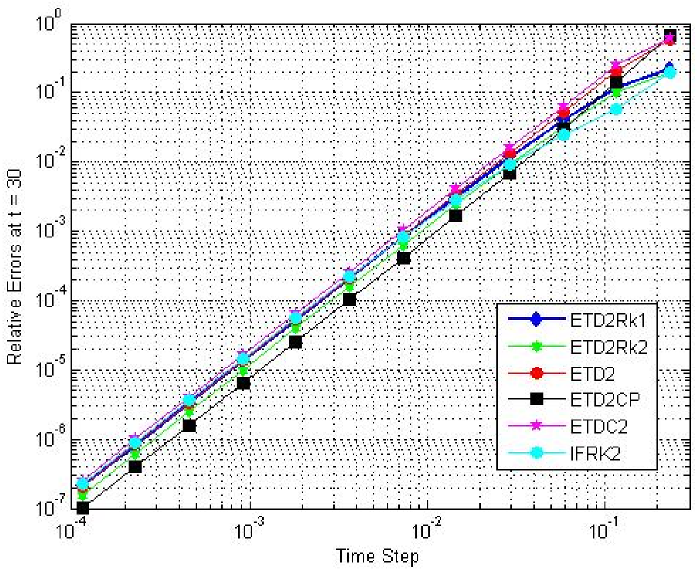


FIGURE 5.2: Time evolution of the numerical solution of the K-S equation (5.18) up to $t = 60$ with the initial condition $w_1(x,0) = \exp(\cos(x/2)),\ x \in [0, 4\pi]$ (5.21).

5.3.1 Computational Results

The results of our experiments are presented in figures 5.3 and 5.5 for the initial condition (5.21), and in figures 5.4 and 5.6 for the initial condition (5.22). In figures 5.3 and 5.4, the numerical relative integrated error (5.17), of using each time discretization method to obtain the numerical solution of the K-S equation (5.18), is plotted as a function of the time step. The exact solution is approximated numerically using $N_{\mathcal{F}} = 64$ grid points in the Fourier spatial discretization. For the time discretization, we use the fourth-order Runge-Kutta method [14] with a very small time-step size. The plots (in figures 5.3 and 5.4) indicate the largest time-step size, i.e. the fewest number of steps, that each method requires to converge to a solution within a fixed given relative error in the figures. The first aspect to emphasize in such figures is that the plots confirm the expected order of the methods. Secondly, a fixed reduction in the time-step size will effectively improve the accuracy, but





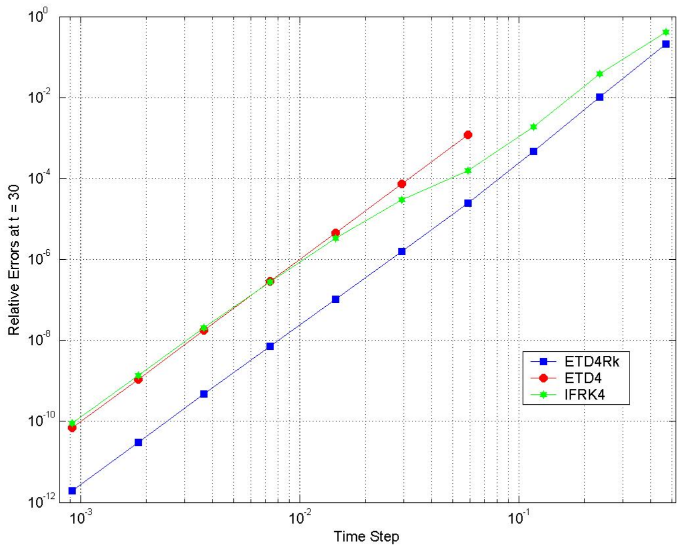
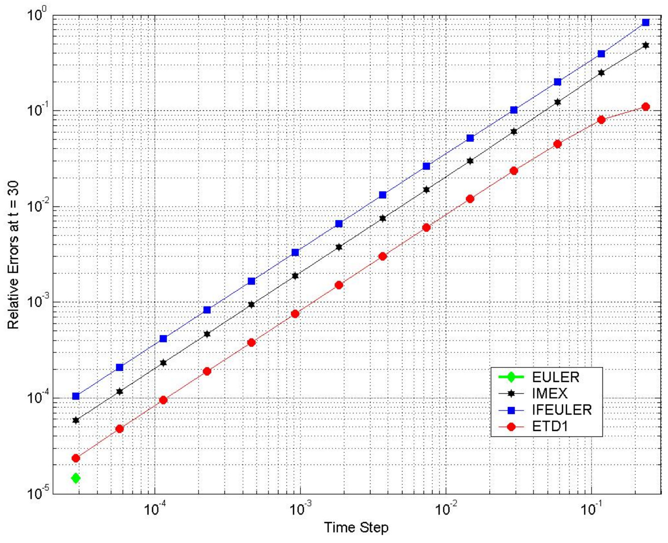
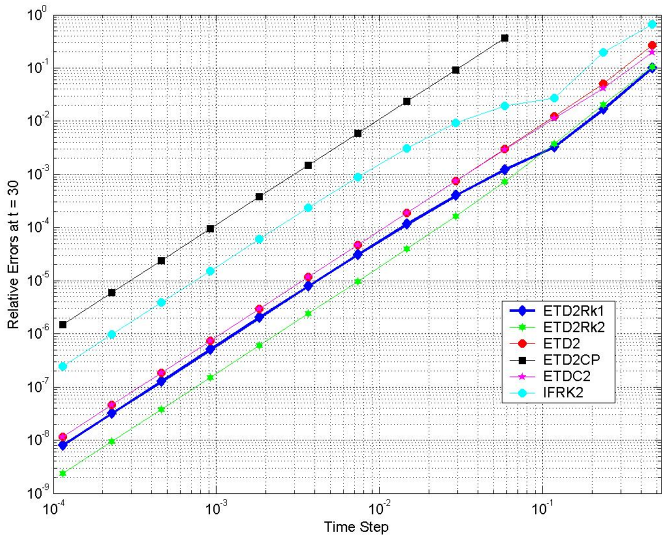


Figure 5.3: Relative errors versus time step for the K-S equation (5.18) with the initial condition $w_1(x,0) = \exp(\cos(x/2)),\ x \in [0, 4\pi]$ (5.21).





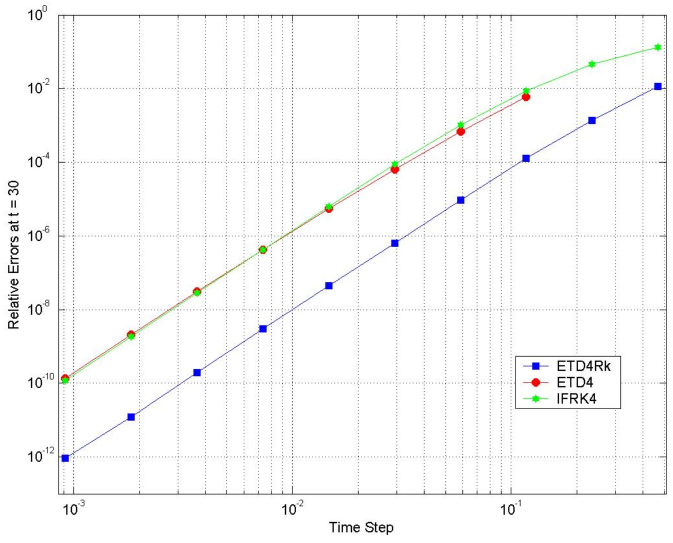


FIGURE 5.4: Relative errors versus time step for the K-S equation (5.18) with the initial condition $w_2(x, 0) = 1.7\cos(\frac{x}{2}) + 0.1\sin(\frac{x}{2}) + 0.6\cos(x) + 2.4\sin(x),\ x \in [0, 4\pi]$ (5.22).

considerably increases the computation cost, which is illustrated in figures 5.5 and 5.6, where we plot the methods' accuracy as a function of the CPU time.

When testing the Euler method (5.3) we find that this method, obviously, requires the smallest number of operations per time step out of all the methods we test. However, because of the numerical stability constraints, the time-step size is limited. Tests show, in figures 5.3 and 5.4, that the Euler method performs well at a very small time-step size, though it breaks down for time-step size larger than $\Delta t \approx 2^{-16}$. This adds to the computation cost as is shown in figures 5.5 and 5.6. Therefore, the Euler method is not in the competition for the "best" method.

Larger time steps may be taken using the other methods that are designed for stiff problems, where there is no such severe restriction for reasons of stability and the time-step size selection is only limited by accuracy.

For the other first-order methods (the ETD1 (5.2), the IFEULER (5.4) and the IMEX (5.5) methods), figure 5.3 indicates that, for the initial condition (5.21), all methods behave similarly and the corresponding errors lie approximately on the same line for all values of the time-step. Furthermore, all methods are inaccurate (errors are of $O(1)$) for time-step sizes larger than $\Delta t \approx 2^{-6}$. On the other hand, figure 5.4 reveals a different behavior of the methods for the initial condition (5.22): it shows a better performance and accuracy of the ETD1 method compared to the other first-order methods. Also, we find that for the time step restriction imposed by the linear term, all methods remain stable at a large value of the time-step $\Delta t \approx 2^{-2}$, but the ETD1 method produces the most accurate solution. Considering the computational cost of the methods, it is clear from figure 5.5 that, for the initial condition (5.21), the methods have an almost identical computation cost per time step. However, in figure 5.6 for the initial condition (5.22), the ETD1 method outperforms the other methods both in speed and in the accuracy of the obtained solution.

For the second-order accurate methods we consider, the ETD2RK1, the ETD2RK2, the ETD2, the ETD2CP, the ETDC2 and the IFRK2 methods. Second-order convergence is confirmed in figure 5.3 for the initial condition (5.21). The performance of all second-order methods is very nearly equivalent here, and the errors for small time steps are almost identical. In addition, the variation in time consumption, for a given level of accuracy, is insignificant, see figure 5.5. However, the ETD2CP method (5.9) slightly outperforms the others both in accuracy and speed referring

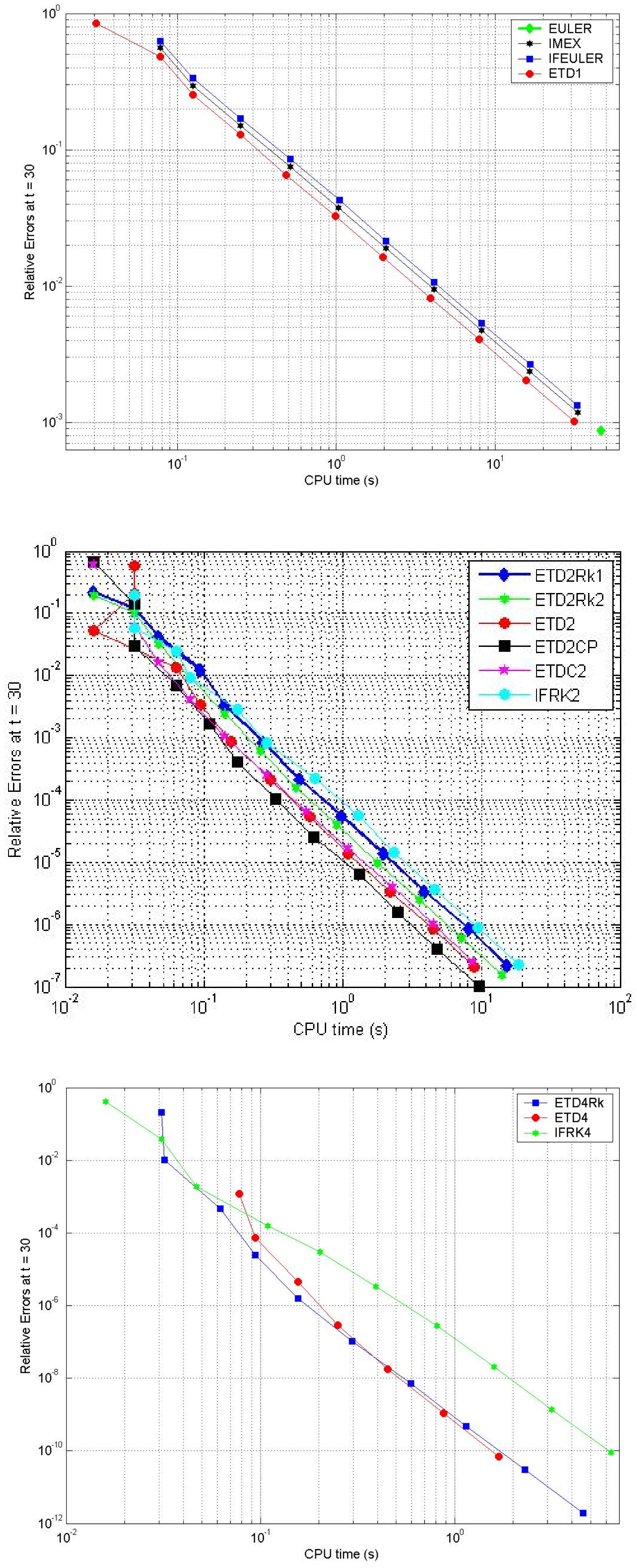
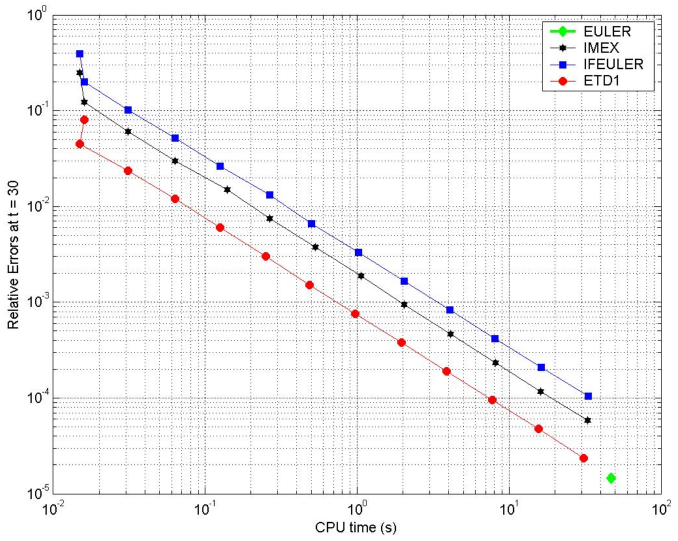
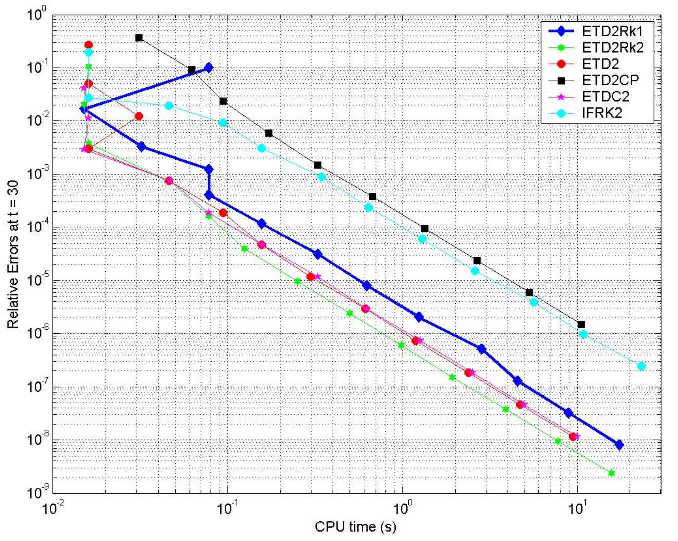


FIGURE 5.5: Relative errors versus CPU time for the K-S equation (5.18) with the initial condition $w_1(x, 0) = \exp(\cos(x/2))$, $x \in [0, 4\pi]$ (5.21).





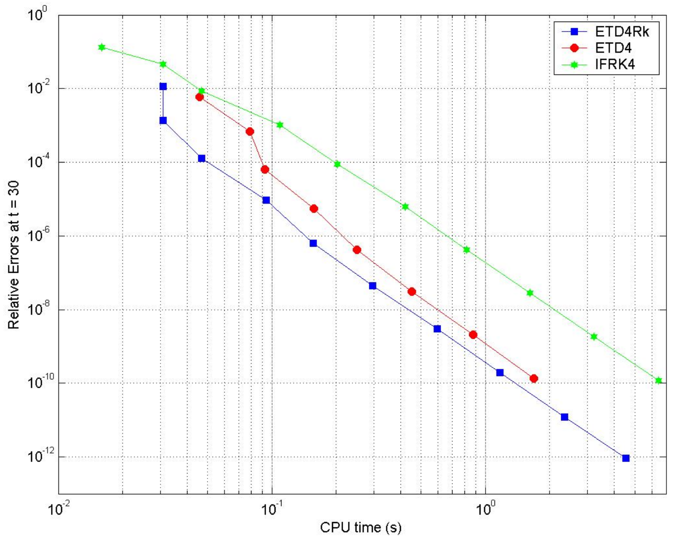


FIGURE 5.6: Relative errors versus CPU time for the K-S equation (5.18) with the initial condition $w_2(x,0) = 1.7\cos(\frac{x}{2}) + 0.1\sin(\frac{x}{2}) + 0.6\cos(x) + 2.4\sin(x),\ x \in [0, 4\pi]$ (5.22).

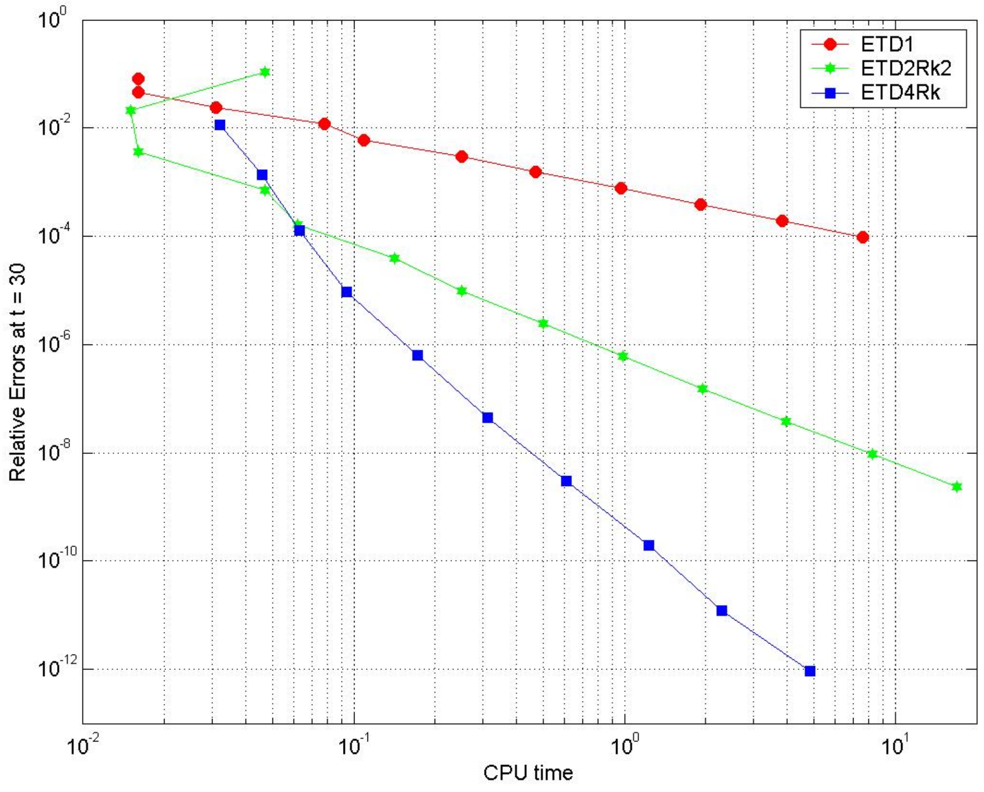


FIGURE 5.7: Relative errors versus CPU time for the K-S equation (5.18) with the initial condition $w_2(x,0) = 1.7\cos(\frac{x}{2}) + 0.1\sin(\frac{x}{2}) + 0.6\cos(x) + 2.4\sin(x)$ (5.22).

to figures 5.3 and 5.5 respectively.

On the other hand, for the initial condition (5.22), we find that of all comparable second-order methods, the ETD2RK2 method (5.8) is the most accurate, for a given time-step size, and the least time consuming for a given level of accuracy, see figures 5.4 and 5.6 respectively. The IFRK2 (5.11) and the ETD2CP methods do not do well for the initial condition (5.22). The ETD2CP method is the least accurate and the most time costly. In addition, figure 5.6 shows that the ETD2 (5.6) and ETDC2 (5.10) methods consume about the same CPU time per time step, while the ETD2RK1 method (5.7) has a longer computation time.

All second-order methods successfully integrate the system for time-step sizes less than $\Delta t \approx 2^{-2}$ and $\Delta t \approx 2^{-1}$ for the initial conditions (5.21) and (5.22) respectively. However, the ETD2CP method fails to be accurate for a larger time-step size than $\Delta t \approx 2^{-4}$ for the initial condition (5.22), see figure 5.4.

For the fourth-order methods, as is evident from figures 5.3 and 5.4, the ETD4

(5.12), the ETD4RK (5.13) and the IFRK4 (5.14) methods behave in a similar way for the two initial conditions. The performance of the IFRK4 method resembles that of the ETD4 method, and the errors for small time steps are almost identical. Clearly the methods have a superior performance, as the accuracy is improved significantly compared to lower order methods. Referring to figures 5.3 and 5.4, the most accurate for a given time step is the ETD4RK method. The ETD4RK method has the advantages (relative to the other fourth-order methods) of being stable for larger time steps[1], having fewer coefficients to evaluate via the Cauchy integral formula approach, and having no starting values to obtain. Also, the ETD4RK method uses slightly less CPU time than the ETD4 method for a given level of accuracy, while the most time-consuming is the IFRK4 method for the two initial conditions (5.21) and (5.22), see figures 5.5 and 5.6.

Second-order time discretization methods have been used often for obtaining numerical solutions for a wide range of PDEs. Reasons for their choice include the difficulties introduced by the combination of nonlinearity and stiffness of a PDE, the increase in complexity both of analysis and implementation for higher-order methods, and in addition, higher-order methods usually require increased computer storage and CPU time. However, when we do a comparison test between the performance of the ETD1 (5.2), the ETD2RK2 (5.8) and the ETD4RK (5.13) methods for solving the K-S equation (5.18) with the initial condition (5.22), we find that the fourth-order method can be very accurate and less costly than the second-order one (the same conclusion was reached by Kassam and Trefethen [44, 45]. The authors found that it is entirely practical to solve nonlinear PDEs to a high accuracy by fourth-order time-stepping methods). In our comparison test we plot, in figure 5.7, the accuracy of these three methods, measured in the relative integrated error (5.17), as a function of CPU time. In the figure, we can see that within the same level of accuracy, the ETD4RK method is less costly than the ETD2RK2 method, whereas the most expensive with high computational cost is the ETD1 method. Thus, the greater accuracy of the ETD4RK and the ETD4 methods more than compensates for the additional computational cost per time step.

[1]This agrees qualitatively with our analysis for the stability region of the ETD4 and the ETD4RK methods in §3.

5.3.2 Conclusion

We have demonstrated how for stiff problems such as the K-S equation (5.18), ETD methods provide an efficient alternative to standard explicit integrators. We have found that the s-step ETD methods (for $s = 1, 2, 4$) all achieve order s and exhibit high accuracy with superior stability properties compared to the explicit method (the Euler method), which imposes a ceiling on the time-step size selection.

To time step our test problem with a second-order method, we can say that, in practice, the consideration of accuracy and computational cost indicates that some of the methods are preferable to others, but all are completely satisfactory. The most efficient choice is the ETD2RK2 method.

Higher order methods are more advantageous. They exhibit higher accuracy and maintain good stability. We have found that the ETD4RK method is marginally the best for the test problem considered.

Regarding accuracy and CPU time in the solving process, we can conclude that the ETD4RK method is clearly favored in most general cases. It is found to be the most stable method with reasonable computational effort. Even at fairly large time-steps, it still maintains good stability and produces high accuracy.

As a final point, this conclusion is limited to the studies of the Kuramoto-Sivashinsky (K-S) equation with the two initial conditions (5.21) and (5.22). The experiments have shown that the performance of the methods varies from one case to the other, and that the ETD and ETD-RK methods of [19] outperform the compared methods for solving the test model (5.18) for the initial condition (5.22). These results cannot be generalized, as they may differ for other choices of initial conditions and for other problems.

5.4 Non-Linear Schrödinger (NLS) Equation

The nonlinear Schrödinger (NLS) equation in one space dimension [10]

$$i\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2} + (V(x) + |u(x,t)|^2)u(x,t), \tag{5.23}$$

where $V(x)$ is the potential function, arises in several different areas of physics, including multi-scale perturbation theory, gravity of electromagnetic waves in a plasma, and the propagation of intense optical light pulses in fibers. The equation gives the wave amplitude $u(x,t)$ as a function of independent variables x (space) and t (time), and it possesses several conservation laws, notably conservation of density, energy and momentum. In addition, it yields a rich variety of nonlinear wave structures, including solitons with arbitrary amplitude and velocity, several kinds of periodic nonlinear wave, and uniform wave-train solutions. The derivation of this equation for the propagation of a plane electromagnetic wave in a nonlinear medium can be found in [10, 42], and an introduction to its mathematical theory is given in [77].

The major application of the NLS equation (5.23) is to the analysis of the propagation of dispersive wave-packets in a nonlinear medium. This equation governs the envelope of wave-packets in the presence of the competing effects of linear dispersion (which tends to smear them out) and nonlinear amplitude dependence (which tends to compress the pulse) of the material properties in a one-dimensional system. When these two competing effects balance, the formation of optical envelope solitons is possible. "Soliton solution" means that the envelope of the nonlinear wave takes the shape of a simple pulse. Solitons are localized waves and are often used to transmit information along optical fibers. They can be ordered in a fashion with the taller solitons moving faster, and the shorter ones moving slower [13]. They also have certain properties, such as clean overtaking of two solitons and clean collisions, i.e. they retain their individual identities (which in addition persist over long distances) after a nonlinear interaction [77].

In our numerical experiments, we use the cubic nonlinear Schrödinger equation

$$i\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2} + |u(x,t)|^2 u(x,t), \tag{5.24}$$

with $V(x) = 0$ in equation (5.23). Equation (5.24) is an example of a problem whose linearization has imaginary eigenvalues, and whose dispersion relation ($\lambda(k) = ik^2$ for wave-numbers k) obtained from a linear stability analysis shows that perturbation to the zero solution neither grow nor decay, but oscillate and travel at speed $-k$. The stiffness in this problem comes from the term $\partial^2 u/\partial x^2$, which results in rapid oscillations of high wave number modes. Transforming equation (5.24) to Fourier space, assuming that the solution satisfies periodic boundary conditions, gives

$$\frac{d\hat{u}_k(t)}{dt} = i(k^2 \hat{u}_k(t) - \mathbf{fft}(|u(t)|^2 u(t))), \tag{5.25}$$

where **fft** is the Matlab command that represents the fast Fourier transform FFT.

We focus primarily on the traveling soliton solutions [10]

$$u(x,t) = a\text{sech}(b(x-vt))e^{i(c_0x+dt)}, \tag{5.26}$$

where $b = \frac{a}{\sqrt{2}}$, $c_0 = -\frac{1}{2}v$ and $d = c_0^2 - b^2$ are real numbers, as explicit exact solutions for the NLS equation (5.24), in testing the efficiency of the first, second and fourth order time discretization methods stated in §5.2. The Euler method (5.3) is never stable for solving the NLS equation for any time-step size. This due to the imaginary eigenvalues of the linearized NLS equation, which are outside the stability region of the method.

Since the exact solutions are known, the numerical results really provide only a check on the numerical methods. The parameters are the speed v and the amplitude a, which are independent. In particular, the larger the velocity v, the more rapid the spatial variation in $u(x,t)$. The amplitude of the wave $u(x,t)$ vanishes at infinity, so, provided we solve on a sufficiently large domain, we can treat the problem as essentially periodic. These solitary waves are known as bright solitons [10]. We direct interested readers to the book by **Billingham** and **King** [10] for a different kind of solitons, known as "dark soliton solutions" for the NLS equation.

Note that, a special case of (5.26), is the non-traveling wave soliton solutions, i.e. $v = 0$, given by

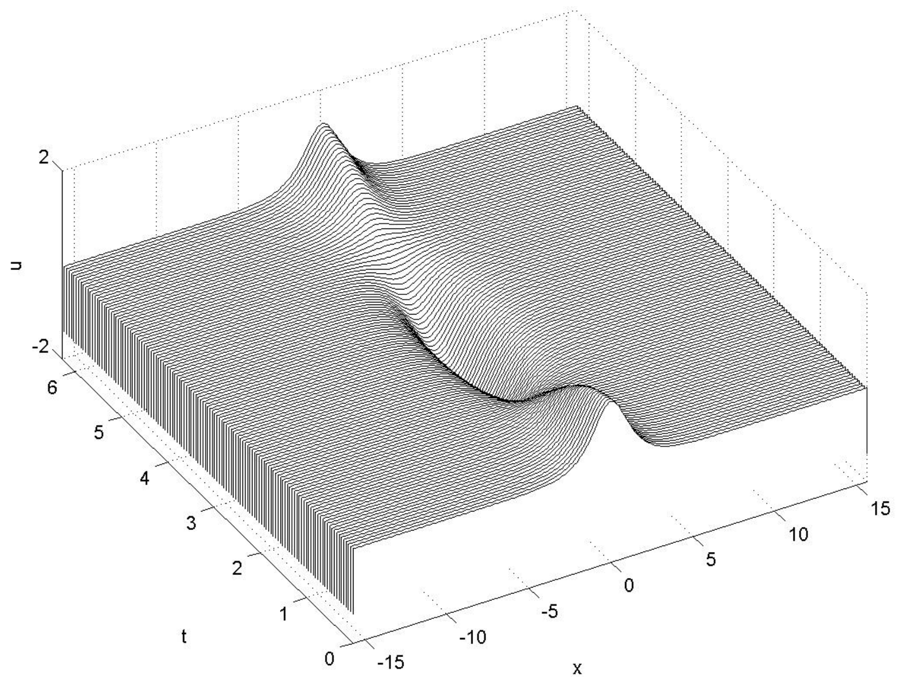
$$u(x,t) = a\text{sech}(bx)e^{idt},$$

with one complete period of soliton oscillation in time $t = 2\pi/|d|$, frequency $d = -b^2$ and amplitude $a = \sqrt{2}b$.

When evaluating the coefficients in the ETD and the ETD-RK methods via the 'Cauchy integral' approach [44, 45] (see §4.2.2), we choose circular contours of radius $R = 1.2$. Each contour is centered at one of the elements that are on the diagonal matrix of the linear part of the semi-discretized model (5.25). The contours are sampled at 32 equally spaced points and approximated by (4.16).

The plots in figure 5.8 give a picture of the solutions (5.26) of the NLS equation (5.24), with the choice of the wave amplitude $a = \sqrt{2}$, $b = 1$ and speeds $v = 0$ and $v = 4$ shown in cases (a) and (b) respectively. For the spatial discretization we use $N_{\mathcal{F}} = 256$ grid points for $x \in [-5\pi, 5\pi]$ (for case (a) $v = 0$) and $x \in [-6\pi, 10\pi]$ (for case (a) $v = 4$). We integrate the system (5.25) using the ETD4RK method (5.13) with time-step size $\Delta t = 2^{-8}$ up to final time $t = 2\pi$ and $t = 6$ for cases (a) and (b) respectively. The solution, in the figure, appears as waterfall plot and shows, in

(a)



(b)

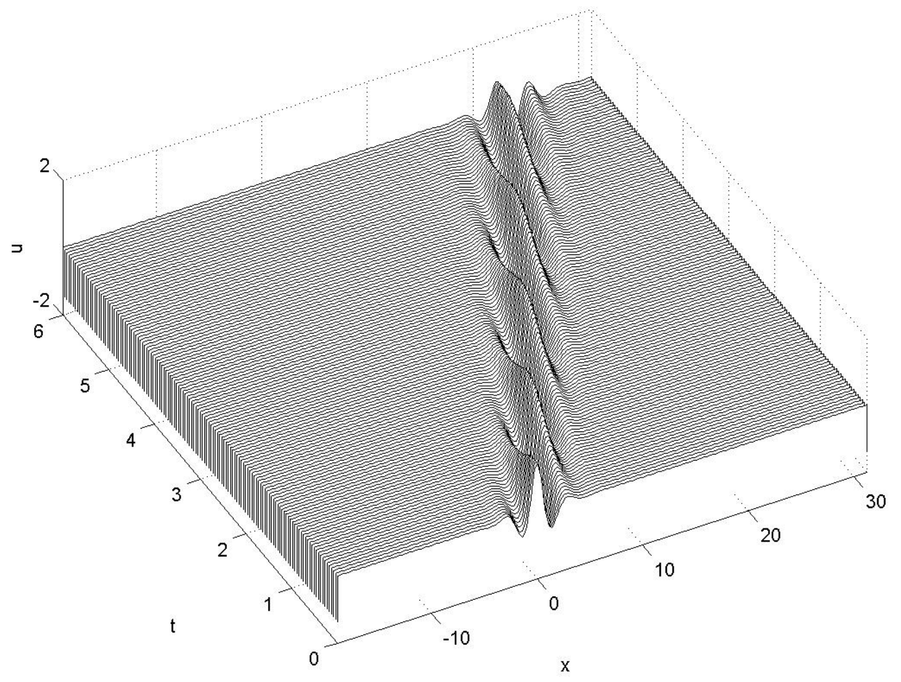


FIGURE 5.8: Real part of the numerical complex solutions (5.26) of the NLS equation (5.24) subject to the initial condition $u(x, t = 0) = \sqrt{2}\text{sech}(x)e^{-ivx/2}$ with (a) speed $v = 0$ and $t = 2\pi$ and (b) speed $v = 4$ and $t = 6$.

case (a), waves oscillating without change in form, while in case (b), the waves are oscillating and traveling with displacements in position.

The results of our experiments in testing the efficiency of the numerical methods, for producing accurate solutions of the NLS equation (5.24), are illustrated in §5.4.1 and §5.4.2 and final conclusions are revealed in §5.4.3.

5.4.1 Computational Results

The comparison results of our experiments are presented in figures 5.9, 5.11 and 5.13, for the first-order, second-order and fourth-order methods stated in §5.2, respectively. We consider the soliton solutions (5.26) as exact solutions for the NLS equation (5.24) and plot, in these figures, the numerical relative integrated error (5.17) of using each time discretization method for solving the test model as a function of the time step. In figures 5.10, 5.12 and 5.14, we plot the accuracy as a function of the CPU time to give an insight into the computation timing. In each figure, we consider integrating the system (5.25) up to final time $t = 6$ for the initial condition

$$u(x, t = 0) = a\mathrm{sech}(bx)e^{-ivx/2}, \ a = \sqrt{2}, \ b = 1, \tag{5.27}$$

with speed $v = 0$ in case (a), while cases (b) and (c), consider the speeds $v = 2, 4$ respectively.

The plots (in figures 5.9, 5.11 and 5.13) confirm the expected order of the methods and indicate the largest time-step size, i.e. the fewest number of steps, that each method requires to converge to a solution within a fixed given relative error in the figures. Improved accuracy is assured if we reduce the time-step size, but this considerably increases the computation cost, which is illustrated in figures 5.10, 5.12 and 5.14), where we plot the methods' accuracy as a function of the CPU time.

For the first and second-order methods, applied to the initial wave (5.27), the spatial discretization is performed using $N_{\mathcal{F}} = 256$ grid points on the space interval $x \in [-5\pi, 5\pi]$ for wave speed $v = 0$. Although the wave's amplitude is not periodic, it vanishes at infinity, so we can still treat the problem as periodic by enlarging the space domain. For the wave with speeds $v = 2, 4$, the solutions travel, and hence, we use $N_{\mathcal{F}} = 512$ grid points on the space interval $x \in [-5\pi, 15\pi]$.

The spatial discretization in the case of utilizing fourth-order methods varies. Generally, an estimate of the total accumulated errors during a time integration consists of the truncation error of the local time step and the rounding errors of the space discretization. Using fourth-order methods with very small time-step size leads to a small truncation error, of $O(\Delta t)^4$, though, with spectral methods the rounding errors are important too. Therefore, to ensure periodicity of the solution and accurate approximation in the space discretization, it is required to enlarge the domain with an increase in the number of grid points. In the case of the initial condition (5.27) with speed $v = 0$, the computations are carried out on a space domain $x \in [-10\pi, 10\pi]$, using $N_{\mathcal{F}} = 512$ grid points. For speeds $v = 2, 4$, we use $N_{\mathcal{F}} = 1024$ grid points on a spatial domain $x \in [-10\pi, 30\pi]$.

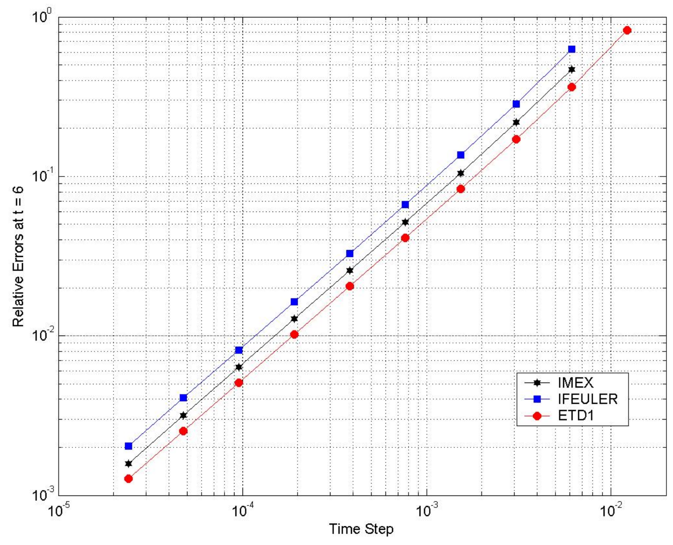
The comparison results for the first-order methods, shown in figure 5.9 (a), indicate that, for the initial condition (5.27) with speed $v = 0$, the best accuracy for a given time-step is achieved by the ETD1 method (5.2), followed by the IMEX method (5.5) (which takes essentially similar CPU times, for a given level of accuracy, as the ETD1 method, see figure 5.10 (a)). The IFEULER method (5.4) requires a smaller time-step and consumes more CPU time than the other comparable first-order methods to produce the same accuracy.

Figure 5.9 (b) shows similar results for the ETD1 method when increasing the speed up to $v = 2$. Clearly the method's performance is superior to the other comparable first-order methods. The method uses the largest time-step size to reach a desired accuracy. The IMEX and the IFEULER methods perform almost identically in achieving the same level of accuracy, with similar CPU times, as illustrated in figure 5.10 (b).

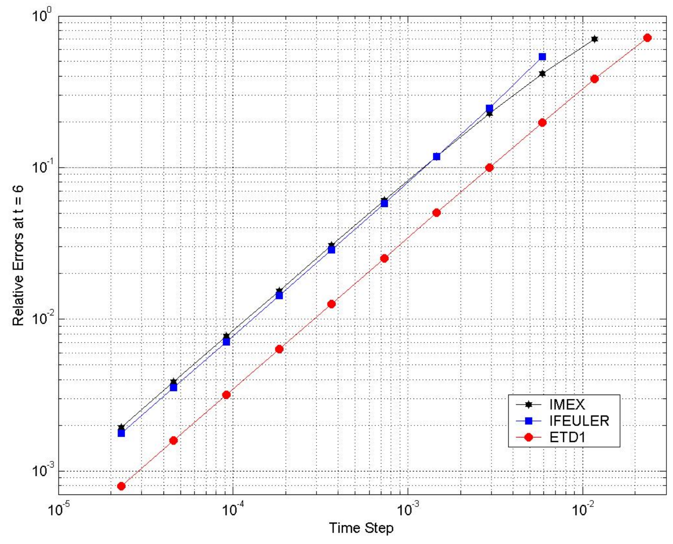
For large speed $v = 4$, the ETD1 method seems to perform slightly worse than the IFEULER method (the most accurate method) for a given time-step size, see figure 5.9 (c). In addition, a considerable variation in CPU times is noticed in figure 5.10 (c) when running the first-order methods on a high-speed ($v = 4$) wave solution. The IFEULER method consumes the least CPU time for a given error, while a large amount is used by the IMEX method. This is due to the fact that a much smaller step size is required for the IMEX method to obtain the same level of accuracy obtained by the other comparable first-order methods.

Further tests are done on the first-order methods for integrating the system at high speeds ($v = 6, 8, \ldots$). In these tests we find that the IFEULER method is the best for producing accurate solutions for the test model (5.24), for a given time-step size. Moreover, increasing the speed v has no effect on its performance, i.e. as the

(a)



(b)



(c)

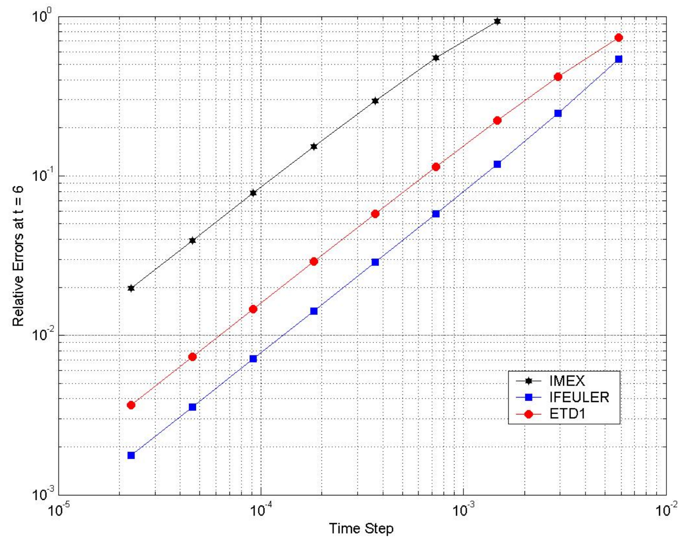
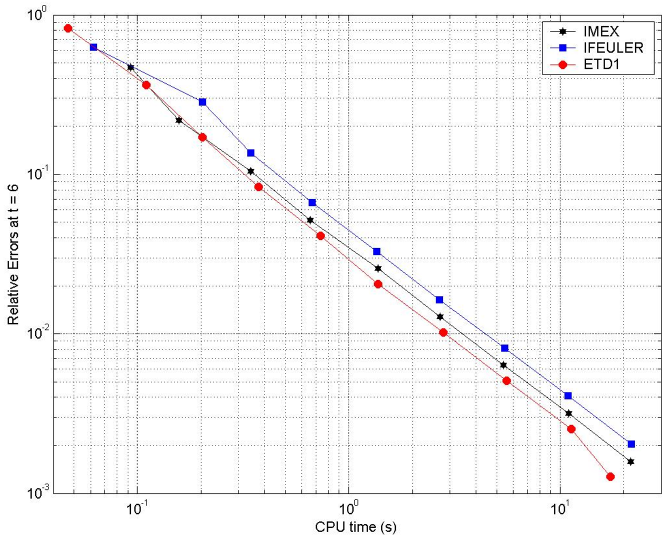
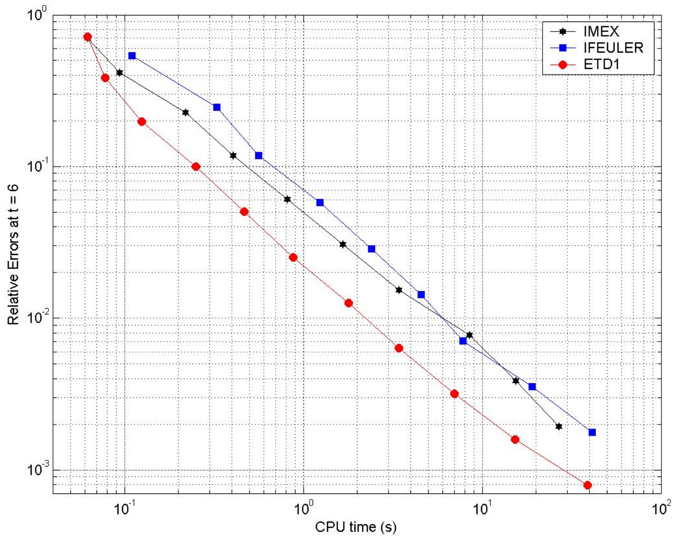


FIGURE 5.9: Relative errors versus time step for (5.24), with the initial condition (5.27), with speeds (a) $v = 0$, (b) $v = 2$ and (c) $v = 4$, for the first-order methods.

(a)



(b)



(c)

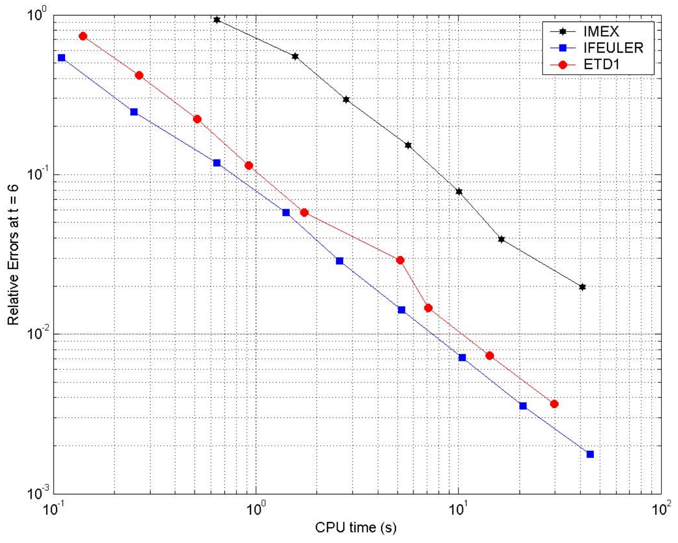
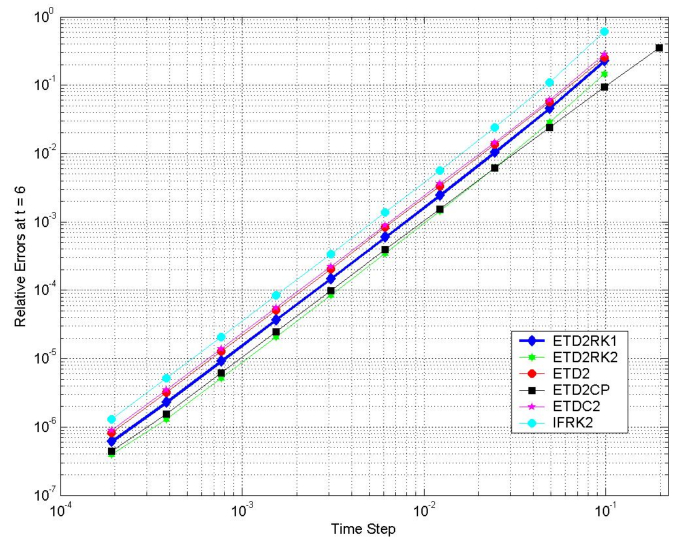
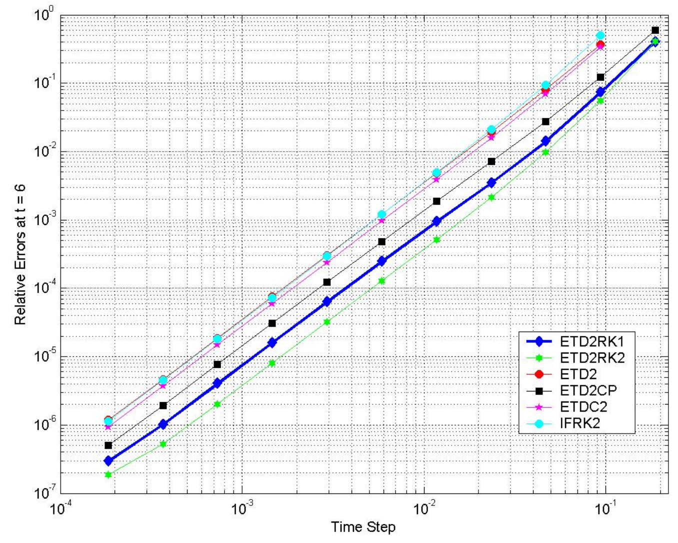


FIGURE 5.10: Relative errors versus CPU time for (5.24), with the initial condition (5.27), with speeds (a) $v = 0$, (b) $v = 2$ and (c) $v = 4$, for the first-order methods.

(a)



(b)



(c)

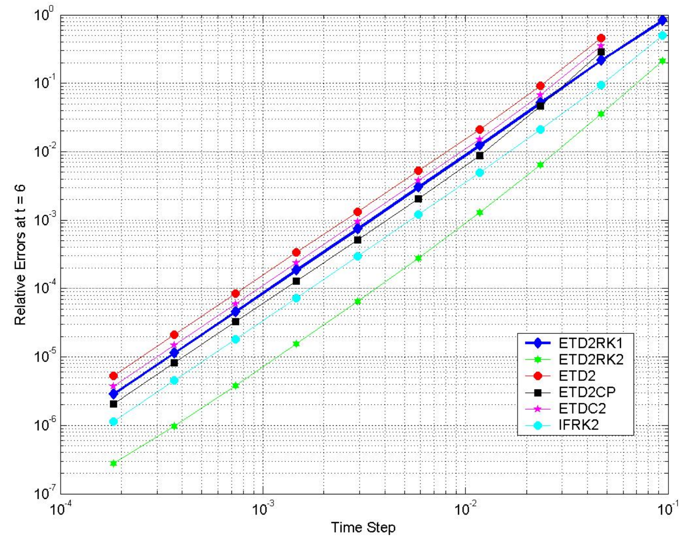
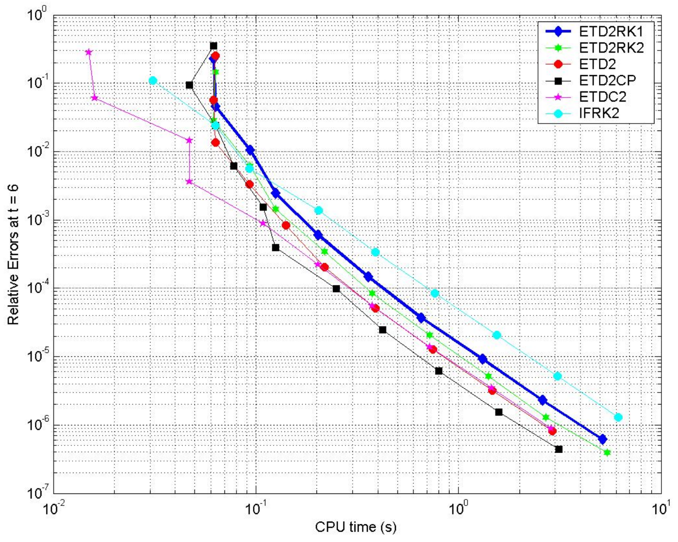
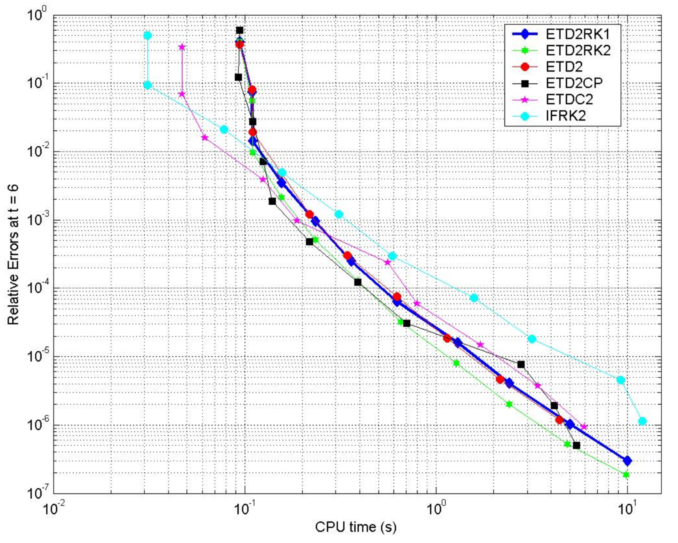


FIGURE 5.11: Relative errors versus time step for (5.24), with the initial condition (5.27), with speeds (a) $v = 0$, (b) $v = 2$ and (c) $v = 4$, for the second-order methods.

(a)



(b)



(c)

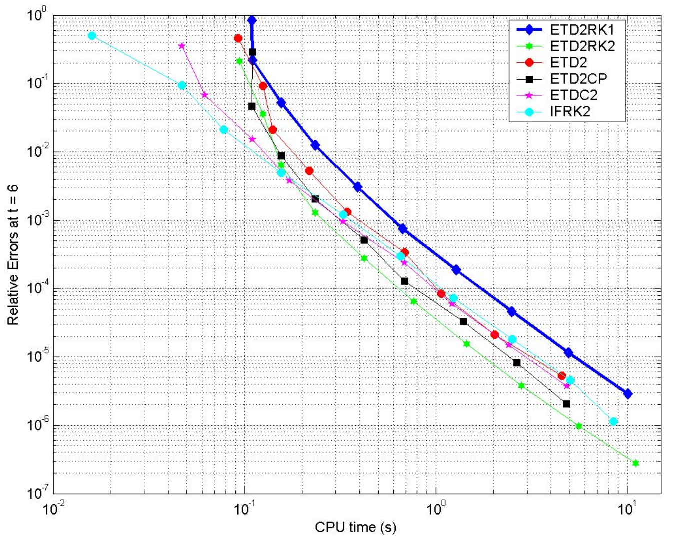
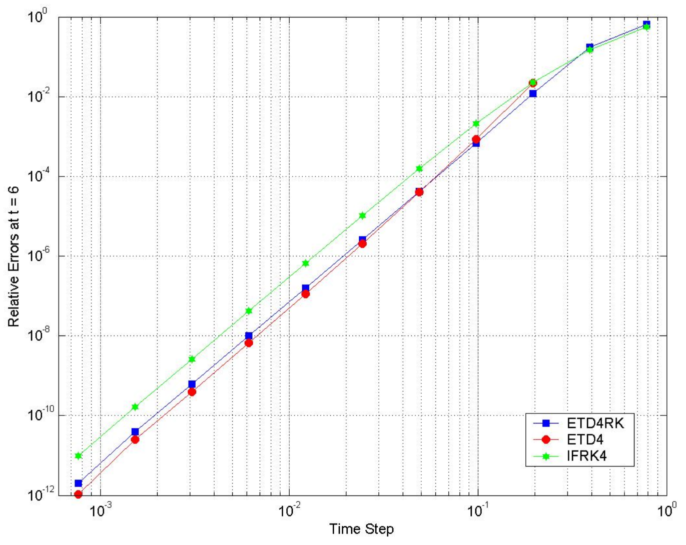
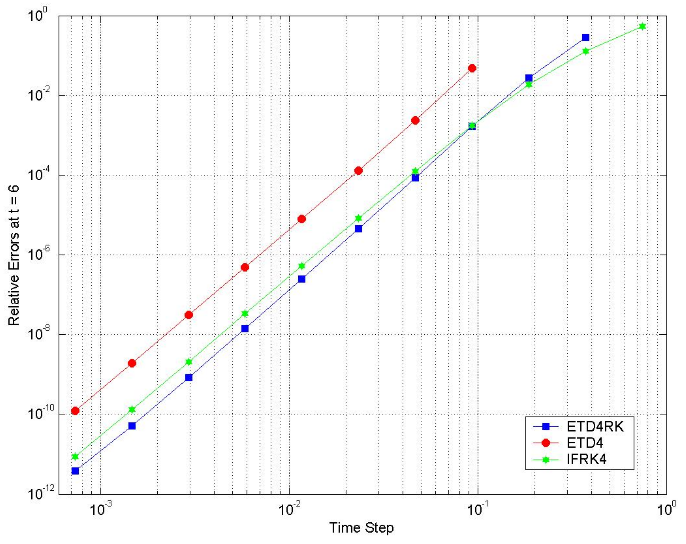


FIGURE 5.12: Relative errors versus CPU time for 5.24), with the initial condition (5.27), with speeds (a) $v = 0$, (b) $v = 2$ and (c) $v = 4$, for the second-order methods.

(a)



(b)



(c)

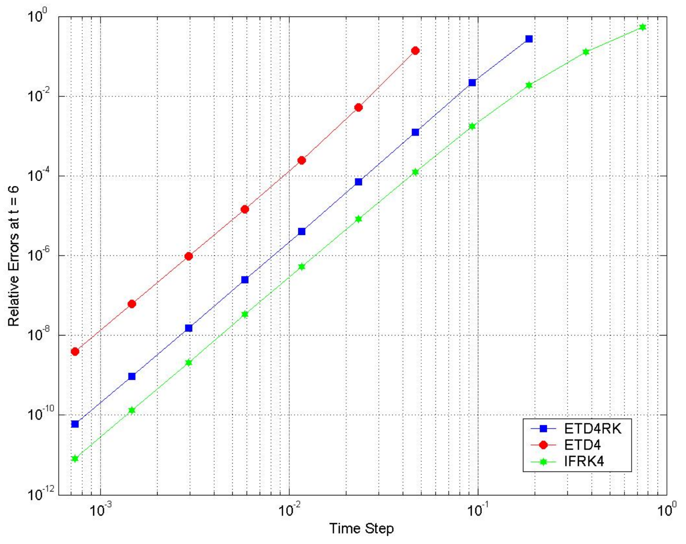
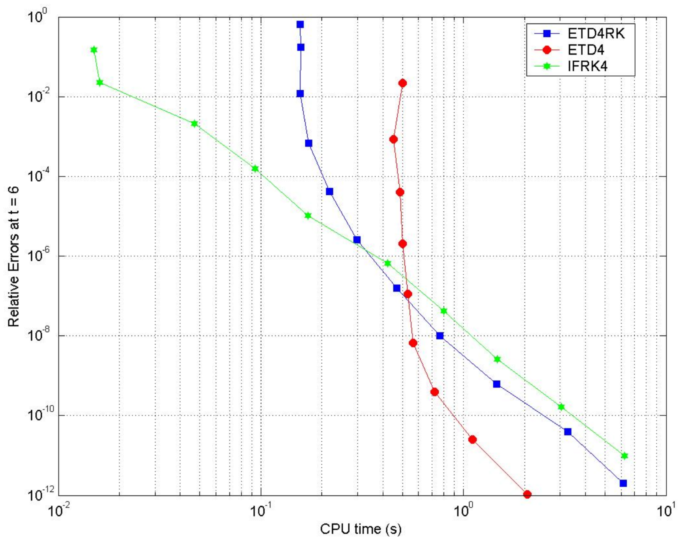
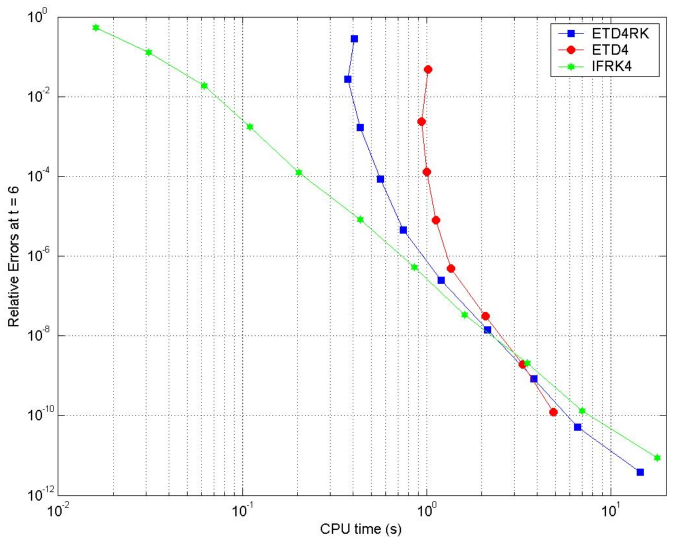


FIGURE 5.13: Relative errors versus time step for (5.24), with the initial condition (5.27), with speeds (a) $v = 0$, (b) $v = 2$ and (c) $v = 4$, for the fourth-order methods.

(a)



(b)



(c)

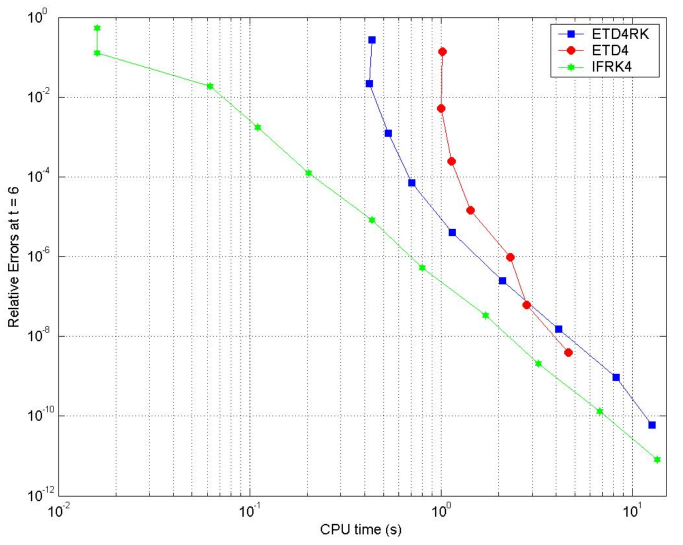


FIGURE 5.14: Relative errors versus CPU time for (5.24), with the initial condition (5.27), with speeds (a) $v = 0$, (b) $v = 2$ and (c) $v = 4$, for the fourth-order methods.

speed increases, we obtain the same quantitative results for the errors over the same range of time-step sizes. This behavior is in contrast with that of the ETD1 and the IMEX methods, which both suffer considerably as the wave speed increases (figures are not shown here as the results resemble those in figure 5.9 (c), for the case of the speed $v = 4$. However, the errors for the ETD1 and the IMEX methods get larger as the speed increases ($v = 6, 8, \ldots$), for the same range of the time-step sizes). The IMEX method needs a much smaller time-step size and much more CPU time to be accurate to the same level obtained by the other first-order competitor methods. In §5.4.2, we further investigate the computational advantages of the IFEULER method and the disadvantages of the ETD1 method.

When integrating the system (5.25) using second-order methods for the initial wave (5.27) with speed $v = 0$, we find that for a given time-step size, the ETD2RK2 method (5.8) is the most accurate and the ETD2 method (5.6) is the third least accurate, while the IFRK2 method (5.11) is the least accurate, see figure 5.11 (a). The performance of the ETD2CP (5.9) and the ETDC2 (5.10) methods is seen to be very similar to that of the ETD2RK2 and the ETD2 methods respectively.

In the case of increasing the speed up to $v = 2$, illustrated in figure 5.11 (b), the performance of the ETD2RK1 method (5.7) resembles that of the previous case ($v = 0$), while that of the ETD2CP method deteriorates. For a given time-step size, the ETD2RK1 method is the second least accurate and the ETD2CP method is third least accurate. The ETD2, the ETDC2 and the IFRK2 methods have almost identical errors and are the least accurate methods for obtaining the numerical solution of the NLS equation (5.24).

Among all comparable second-order methods, the ETD2RK2 shows the best performance overall (in cases of $v = 0, 2, 4, 6$) and is the most accurate method for a given time-step size (see figure 5.11 (c) for the case of $v = 4$, although errors are equivalent to that of the IFRK2 method in the case of $v = 6$ (not shown)). Increasing the speed up to $v = 4, 6, \ldots$ has a significant effect in reducing the performance of the ETD2RK1, the ETDC2, the ETD2 and the ETD2CP methods. See the case of $v = 4$ in figure 5.11 (c), where these methods require a smaller time-step size than that used by the ETD2RK2 method to obtain a desired accuracy. On the other hand, increasing the speed has no impact on the performance of the IFRK2 method. This method is seen to have the same quantitative error for a given time-step size as the speed increases, see figure 5.11 (a), (b) and (c). For larger speeds

$v = 6, \dots$ and for a given time-step size, the IFRK2 method is the most accurate method.

Regarding CPU time consumption, figure 5.12 reveals that, generally, the variations in CPU time between the comparable second-order methods are insignificant (when required to be accurate to some specified level). However, figure 5.12 shows that, in cases (a) $v = 0$ and (b) $v = 2$, the IFRK2 method is the most expensive in timing, as it uses a smaller time-step size than that used by the other second-order methods to reach a desired level of accuracy. In case (c) $v = 4$, we find that the ETD2RK2 method achieves a required accuracy level fastest, while the ETD2RK1 is the most time consuming.

We finally consider the performance of the fourth-order methods. It can be seen in figure 5.13 that all methods show a clear fourth-order behavior. For speed $v = 0$ in the initial condition (5.27) (figure 5.13 (a)), the IFRK4 method (5.14) is slightly less accurate, for a given time-step size, than the ETD4 (5.12) and the ETD4RK (5.13) methods, which show the best performance in producing accurate numerical solutions (errors are seen to be very similar for a given time step).

As the speed increases $v = 2, 4, \dots$, we find that the IFRK4 method has the same accuracy for a given time-step size (this is similar to the behavior of the IFEULER and the IFRK2 methods. The analysis applied to explain the behaviors of the IFEULER method's performance in §5.4.2, can also be used to analyze the performance of the IFRK2 and the IFRK4 methods). In addition, the performance of the ETD4RK method deteriorates gradually as v increases, while that of the ETD4 method deteriorates dramatically (see figure 5.13 (b) and (c), of which the case of speed $v = 4$ (c) shows that, for a given time-step size, the IFRK4 method is most accurate for obtaining the numerical solution).

In figure 5.14, we find that the exponential integrators rely on the fast evaluation of the exponential and related functions. The computations of the ETD and the ETD-RK methods coefficients (which are done once at the beginning of the integration for each time-step size) have a noticeable effect on the CPU times, as they impose a significant timing overhead when the methods use large time-step size. As can be seen in figure 5.14 (a), (b) and (c), the differences in timing between fourth-order methods, for large time-step size, become significant for a required level of accuracy. In general, the ETD4 method is the most computationally expensive, since most of the CPU time is spent on setting up the coefficients, via the complex integration (four more coefficients than the ETD4RK method). In addition, as we decrease the time-step size, the variation in the time consumed by ETD4 and ETD4RK methods is negligible for a considerable range of specified error tolerances. However, for small time-step, and for non-traveling solitons $v = 0$, the ETD4 method in fact consumes least CPU time, as it can use the largest time-step size to reach a desired accuracy. The IFRK4 method is marginally the slowest for $v = 0$, as shown in figure 5.14 (a). As the speed increases $v = 2, 4$, see figure 5.14 (b) and (c) respectively, the IFRK4 method maintains its rapidity in accomplishing the computations, whereas the computation times of the ETD4 and the ETD4RK methods increase gradually. For higher speeds these methods use a smaller time-step size to obtain a level of accuracy similar to that obtained for $v = 0$.

We note finally that, for accurate and economical computations, it is often advantageous to utilize fourth-order methods. The benefits of these methods are that they can use a much larger time-step size than that for the lower order comparable integrators for an equivalent level of accuracy, and hence they are cheap.

5.4.2 Error Analysis of the ETD and the IF Methods

Our analysis in this section investigates the computational advantage of the Integrating Factor IF methods and the disadvantage of the Exponential Time Differencing ETD methods in solving the nonlinear Schrödinger (NLS) equation (5.24). The previous tests in §5.4.1 revealed that, increasing the speed $v = 0, 2, \ldots$ of the soliton solutions (5.26) has no effect on the IF methods' performance, i.e. as the speed increases, we obtain the same quantitative results for the errors over the same range of the time-step sizes for an s-order IF method. This behavior is in contrast with that of the ETD methods, which suffer considerably as the wave speed increases, and so, in cases of large speeds $v = 4, 5, \ldots$ and for a given time-step size, the IF methods are the best for producing accurate soliton solutions.

The usual way to analyze the accuracy of a numerical method is to study the local truncation error in time. So if we consider the model (5.1)

$$\frac{du(t)}{dt} = cu(t) + F(u(t), t),$$

then the local truncation error, for example, of the ETD1 (5.2) and the IFEULER

(5.4) methods are

$$L.T.E_1 \approx \frac{\Delta t^2}{2} dF(u(t),t)/dt, \tag{5.28a}$$

$$L.T.E_2 \approx \frac{\Delta t^2}{2} d(F(u(t),t)e^{-ct})/dt, \tag{5.28b}$$

respectively. The derivation of equations (5.28a) and (5.28b) is given in Appendix B. For cases where the nonlinear term $F(u(t),t)$ is slowly varying, i.e. the value of $dF(u(t),t)/dt$ is small, the ETD1 method is highly accurate, while in cases where the term $F(u(t),t)e^{-ct}$ is slowly varying ($d(F(u(t),t)e^{-ct})/dt$ is small), the IFEULER is the most accurate.

To understand clearly how this analysis can be applied to the numerical methods in case of the soliton solutions (5.26), we need first to apply the analysis to simple exact periodic solutions of the NLS equation (5.24) of the form

$$u(x,t) = Ae^{i(\omega t + Cx)},$$

with amplitude A, and frequency $\omega = C^2 - A^2$.

According to (5.28)

$$L.T.E_1 \approx \Delta t^2 \omega A^3 e^{i(\omega t + Cx)}/2, \qquad L.T.E_2 \approx -\Delta t^2 A^5 e^{i(Cx - A^2 t)}/2.$$

Hence, $L.T.E_1 \propto \omega A^3 = (C^2 - A^2)A^3$ and $L.T.E_2 \propto A^5$. For cases in which ω is small, i.e. C and A are quantitatively similar, the ETD1 method (5.2) should therefore show the best accuracy for a given time-step size, while if A is small and C is large, i.e. ω is large, the IFEULER method (5.4) should be the best.

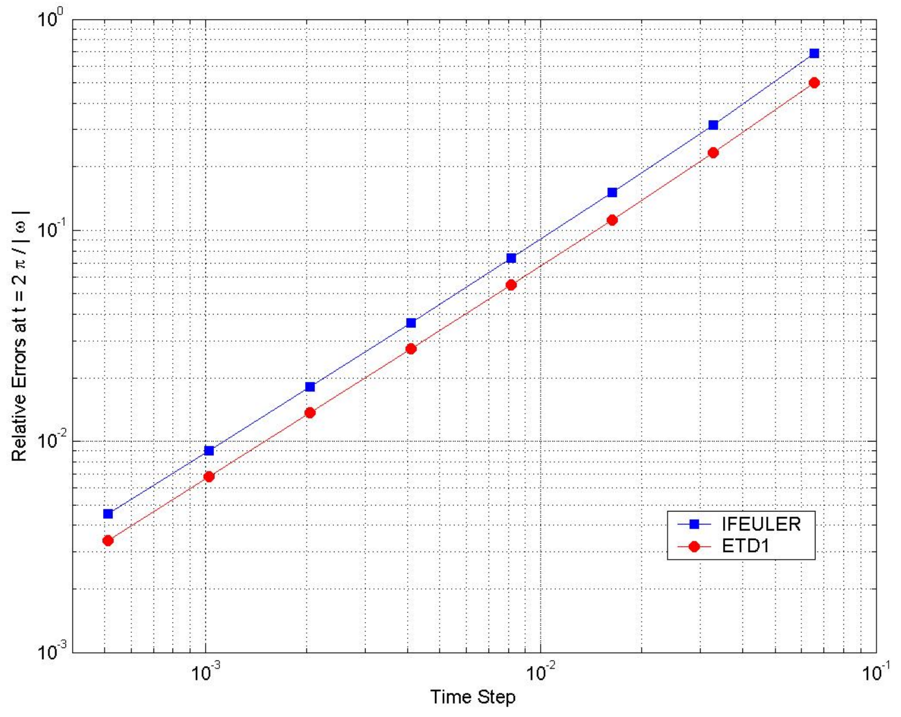
In figure 5.15, we plot the numerical relative error of the integrated error norm (5.17) as a function of the time-step size, for two different values of ω. The choices of our exact solutions are

$$\begin{aligned} u_1(x,t) &= e^{i(-3t/16 + x/4)}/2, \ x \in [0, 8\pi], \\ u_2(x,t) &= e^{i(63t + 8x)}, \ x \in [0, \pi/4]. \end{aligned}$$

We use $N_{\mathcal{F}} = 64$ grid points for the space discretization, and integrate the system (5.25) up to one period of time $t = 2\pi/|\omega|$, utilizing the ETD1 (5.2) and the IFEULER (5.4) methods. We evaluate the coefficients in the ETD1 method using the 'Cauchy integral' approach. Figure 5.15 case (a) confirms that, for the initial condition

$$u_1(x,0) = e^{ix/4}/2,$$

(a)



(b)

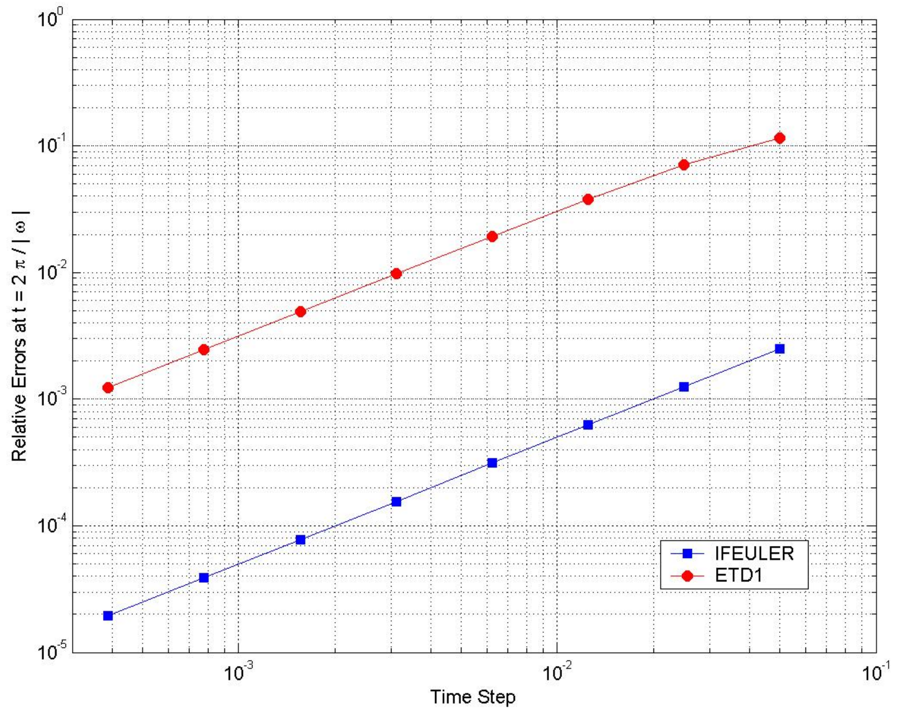


FIGURE 5.15: Relative errors versus time step for solving the NLS equation (5.24) with the ETD1 (5.2) and the IFEULER (5.4) methods, subject to the initial condition (a) $u_1(x,0) = e^{ix/4}/2,\ x \in [0, 8\pi]$ (b) $u_2(x,0) = e^{i8x},\ x \in [0, \pi/4]$.

with the small value of $\omega = -3/16$, the ETD1 method gives the best accuracy for a given time-step size, while, case (b) of the figure shows that for the initial condition

$$u_2(x,0) = e^{i8x},$$

with the small value of $A = 1$ and large value of $\omega = 63$, the IFEULER method is the most accurate method for solving the NLS equation (5.24).

The above analysis can also be applied in a similar way to the soliton solutions

$$u(x,t) = a\text{sech}(b(x - vt))e^{i(c_0x+dt)}, \tag{5.29}$$

of the NLS equation (5.24)

$$\frac{\partial u(x,t)}{\partial t} = -i\Big(\frac{\partial^2 u(x,t)}{\partial x^2} + |u(x,t)|^2 u(x,t)\Big).$$

For these solutions, the non-linear part takes the form

$$F(u(t),t) = -i|u|^2 u = -ia^3 \text{sech}^3\big(a(x - vt)/\sqrt{2}\big)e^{i(-vx/2+(v^2/4-a^2/2)t)}, \tag{5.30}$$

and the linear operator $(-i\partial^2 u/\partial x^2)$ corresponds to the term $c = ik^2$ for waves with wave-number k. If we look again at (5.28a), we find that if the soliton solutions (5.29) vary slowly in time, then the non-linear part $F(u(t),t)$ (5.30) also varies slowly in time, and the ETD1 method (5.2) is highly accurate. As we increase the speed v, however, the value of $dF(u(t),t)/dt$ increases, and the ETD1 method becomes less accurate (this was seen in our earlier experiments illustrated in figure 5.9 (c) for the case of the speed $v = 4$). On the other hand, if we look at (5.28b) again for the non-linear term $F(u(t),t)$ (5.30), we find that if the term $F(u(t),t)e^{-ct}$ varies slowly in time, then the IFEULER method (5.4) should be the most accurate for a given time step. Here, as we vary the value of the speed v, the value of $d(F(u(t),t)e^{-ct})/dt$ does not change, and therefore, the IFEULER method obtains the same quantitative results for the errors over the same range of time-step sizes (this is illustrated in figure 5.9 (a) $v = 0$, (b) $v = 2$ and (c) $v = 4$).

To give a more evident view of the above analysis, fix $a = \sqrt{2}$ (as in our earlier experiments in §5.4.1) and assume that the dominant wave-number is approximately $k = -v/2 \Rightarrow c = iv^2/4$, and hence for the nonlinear term $F(u(t),t)$ (5.30)

$$dF(u(t),t)/dt \propto (v^2/4 - a^2/2)a^3.$$

Therefore, as the speed v increases, the nonlinear term $F(u(t),t)$ varies rapidly and the value of $dF(u(t),t)/dt$ increases, and according to (5.28a), the ETD1 method

(5.2) become less accurate than the IFEULER method (5.4) for a given time step. We would expect the ETD1 method to be highly accurate when $v = 0$ and $v^2/4 = a^2/2$, i.e. $v = 2$, which agrees with our results in figure 5.9 (b). On the other hand the term

$$F(u(t), t)e^{-ct} = -ia^3 \text{sech}^3(a(x - vt)/\sqrt{2})e^{i(-vx/2 - a^2t/2)},$$

and hence, the value $d(F(u(t), t)e^{-ct})/dt$ is not influenced by increasing the speed v, and the truncation error (5.28b) does not change and therefore the IFEULER method obtains the same quantitative results for the errors over the same range of time-step sizes.

Finally, we note that the above investigation can also be useful to explain the behaviors of the second and fourth-order ETD and IF methods in solving the NLS equation (5.24), illustrated in figures 5.11 and 5.13 in our earlier experiments.

5.4.3 Conclusion

We have implemented several competing exponential integrators for a large stiff system of ODEs arising from the space discretization of the NLS equation (5.24). Our simulations for soliton solutions have revealed considerably different performances of the compared numerical methods in different cases, which make it clear that the best choice of method depends on the specific problem to be solved. However, all compared methods have been able to resolve oscillatory solitons to a required error tolerance without the severe time-step size restrictions of the standard schemes.

Experimentally, we have found that, in the non-traveling wave soliton solution case ($v = 0$ in (5.26)), the most efficient methods of those we have compared are the ETD and ETD-RK methods of **Cox** and **Matthews** [19]. The best of these methods are the first-order ETD1 (5.2), the ETD2RK2 (5.8) and the ETD4RK (5.13) methods. Similar conclusions have been found in the case of slowly traveling soliton solutions (i.e. the speed $v = 2$ in our tests).

In addition, we have found firstly that, the performance of IFEULER (5.4), IFRK2 (5.11) and IFRK4 (5.14) methods is not influenced by varying the values of the speed v, i.e. if we compare the performance of similar order IF methods, as the speed increases, we get the same quantitative results for the errors over the same range of time-step sizes. Secondly, to produce accurate numerical solutions of the NLS equation for larger speeds v, the IF methods prove to be the most accurate.

5.5 Thin Film Equation

As a next step towards solving nonlinear PDEs, we include the fourth-order thin film equation [36]

$$\frac{\partial\mathcal{H}(x,t)}{\partial t}=-\frac{\partial\mathcal{H}(x,t)}{\partial x}+\frac{\partial}{\partial x}\Big(\mathcal{H}^3(x,t)\Big(\gamma\cos x-\alpha\Big(\frac{\partial^3\mathcal{H}(x,t)}{\partial x^3}+\frac{\partial\mathcal{H}(x,t)}{\partial x}\Big)\Big)\Big),\quad(5.31)$$

where the film thickness $\mathcal{H}(x,t)$ is a 2π-periodic function. The authors of [36] studied the time-dependent evolution equation (5.31) for a free-surface of a thin film viscous fluid flow exterior to a rotating horizontal circular cylinder in a vertical gravitational field with a polar angle x of a point on the cylinder. The model is based on a lubrication approximation assuming that the film is very thin compared to the cylindrical radius, and includes the effect of cylindrical rotation $(-\partial\mathcal{H}/\partial x)$, gravity $(\partial(\gamma\mathcal{H}^3\cos x)/\partial x)$ and surface-tension $(-\partial(\alpha\mathcal{H}^3(\partial^3\mathcal{H}/\partial x^3+\partial\mathcal{H}/\partial x))/\partial x)$ with corresponding parameters γ and α. The theoretical analysis and physical interpretation of equation (5.31) in [36] revealed that distinct physical mechanisms, governing a slow approach to a steady state, occur on different time-scales. Firstly, there is the fast process of rotating with the cylinder. Secondly, surface-tension squeezes the free fluid surface to a cylindrical shape. After this, oscillations decay exponentially on a slow time scale. Their numerical investigation revealed that the solution oscillates with time before eventually decaying to a steady state at large time.

A very brief list of some recent research in fluid dynamics involving numerical simulations of this class of problems includes [6, 24, 88]. The authors of [6] obtained an evolution equation and analyzed the dynamics of a thin viscous film which lines a rigid cylindrical tube and surrounds a core of inviscid fluid considering flow in the 2D cross section of the tube. When solving the full nonlinear system numerically, they found that the film can evolve towards a steady solution of uniform thickness. In addition, a model for the evolution of a thin liquid film flowing on and coating a horizontal cylinder that is rotating uniformly about its axis is presented in [24]. The authors of [24] obtained solutions to the evolution equation with implicit numerical schemes based on finite differences. The results showed a wide range of possible behavior depending on the rotation rate.

Solving the thin film equation (5.31) numerically is a challenging task since the numerical solution poses several problems:

1. The fourth-order term is very stiff: the stability constraint on the time step for explicit methods requires $\Delta t \approx O(h^4)$ for space step h.

2. When integrating the semi-discretized system of the equations, we find that

   - Applying fully implicit methods requires at each time step the solution of a system of nonlinear equations.
   - Applying the IMEX methods requires at each time step the calculation of either a pentadiagonal differentiation matrix inverse (in the case of discretizing with finite difference approximations [24]) or a full dense differentiation matrix inverse (in the case of a Fourier spectral approximation). For example, applying a first-order IMEX method gives

     $$\mathcal{H}_{n+1} = \cdots + (I + \alpha \Delta t \mathcal{H}_n^3 D_4)^{-1} \mathcal{H}_n + \cdots ,$$

     where I is the identity matrix, D_4 is the corresponding differentiation matrix for the fourth derivative and $\mathcal{H}_n$ denotes the numerical approximation to $\mathcal{H}(t_n)$.
   - We cannot apply exponential integration methods directly to solve such problems since these methods are designed for PDEs that can be split into linear and nonlinear parts.

To facilitate numerical studies of the thin film equation (5.31), we set the perturbation

$$\mathcal{H}(x,t) = h_0 + u(x,t),$$

where h_0 is the mean film thickness, to be the solution of the equation and obtain

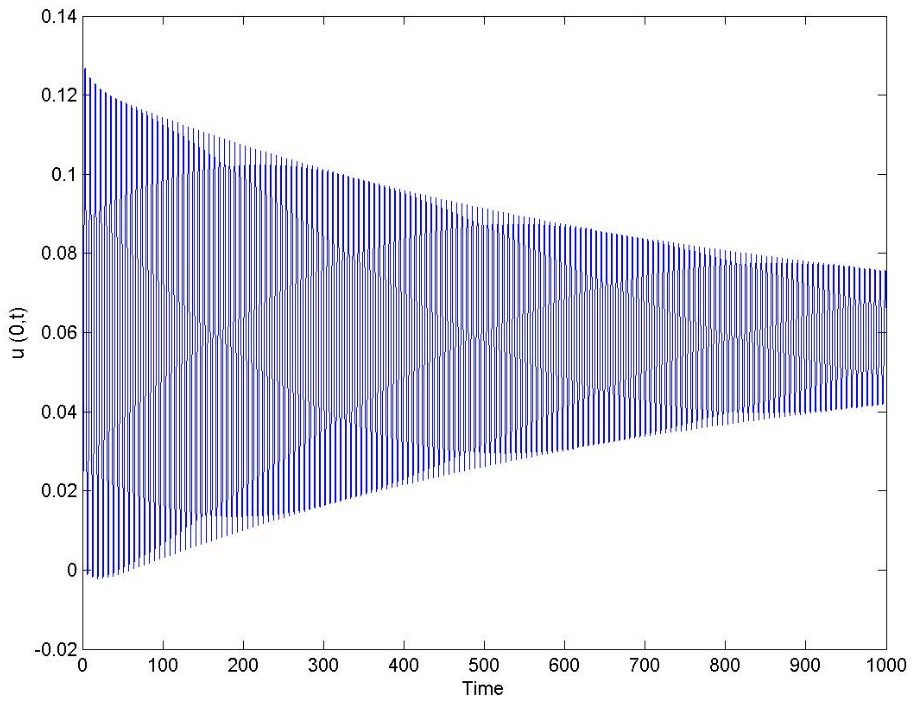
$$\frac{\partial u(x,t)}{\partial t} = -\frac{\partial u(x,t)}{\partial x} + \frac{\partial}{\partial x}\Big((h_0 + u(x,t))^3\Big(\gamma \cos x - \alpha\Big(\frac{\partial^3 u(x,t)}{\partial x^3} + \frac{\partial u(x,t)}{\partial x}\Big)\Big)\Big).$$

After some algebraic manipulation to split the linear and nonlinear terms we deduce

$$\begin{aligned}\frac{\partial u(x,t)}{\partial t} &= -\frac{\partial u(x,t)}{\partial x} - \alpha h_0^3\Big(\frac{\partial^4 u(x,t)}{\partial x^4} + \frac{\partial^2 u(x,t)}{\partial x^2}\Big) - \gamma h_0^3 \sin x \\ &\quad + \frac{\partial}{\partial x}\Big(((h_0 + u(x,t))^3 - h_0^3)\Big(\gamma \cos x - \alpha\Big(\frac{\partial^3 u(x,t)}{\partial x^3} + \frac{\partial u(x,t)}{\partial x}\Big)\Big)\Big), \quad (5.32)\end{aligned}$$

which is an exact reformulation of the original thin film equation (5.31). The perturbation $u(x,t)$ is either small, leading to a weakly nonlinear PDE or large, leading to a strongly nonlinear PDE.

(a)



(b)

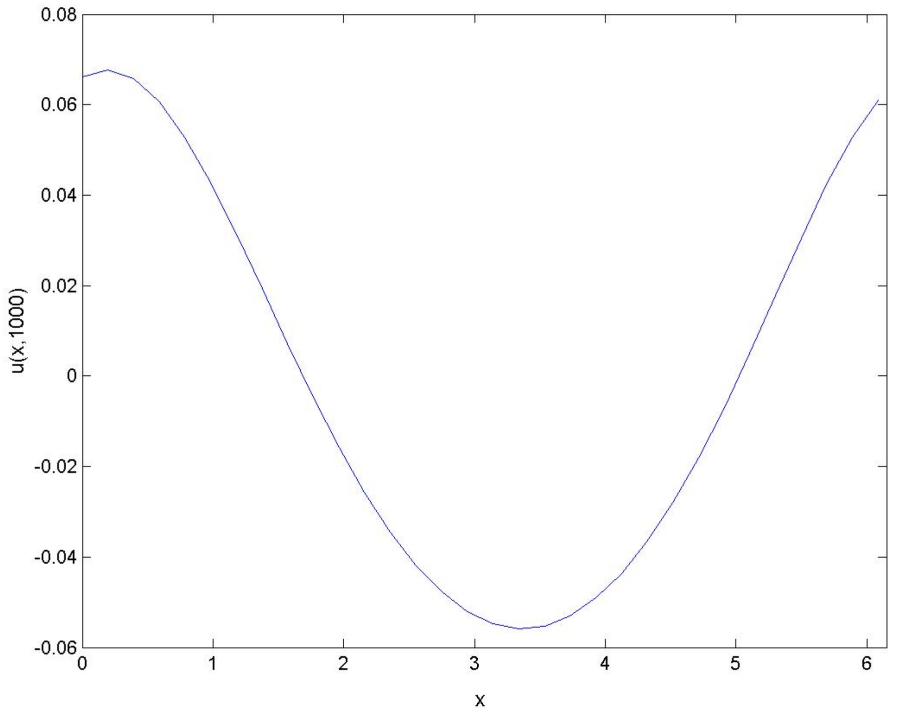


FIGURE 5.16: Time evolution of the thin film equation (5.32) with $\alpha = 0.0048, \gamma = 0.0532$ and initial film thickness $\mathcal{H}(x, 0) = 1$ (a) solution at polar angle $x = 0$, in the time interval $0 \leq t \leq 1000$ (b) an approach to a steady state at $t = 1000$.

Discretizing the spatial derivatives of the thin film equation (5.32), with periodic boundary conditions, in Fourier space yields

$$\begin{aligned}\frac{d\hat{u}(t)}{dt} &= (-ik + \alpha h_0^3(k^2 - k^4))\hat{u}(t) - \gamma h_0^3\mathbf{fft}(\sin x) \\ &\quad + ik[\mathbf{fft}((\gamma\cos x + i\alpha\Re(\mathbf{ifft}((k^3 - k)\hat{u}(t))))((u(t) + h_0)^3 - h_0^3))], \quad (5.33)\end{aligned}$$

where k is the wavenumber and **fft** and **ifft** are Matlab commands that represent the fast Fourier transform FFT and its inverse respectively. The FFT is a complex transform, but in most applications, the data u to be differentiated is real, hence, only the real part $\Re$ is taken in the spectral differentiation. The semi-discrete system (5.33) is a system of coupled ODEs in time. The stiffness in the system is due to the fact that the diagonal linear operator, with elements $-ik + \alpha h_0^3(k^2 - k^4)$, has complex eigenvalues of which some have large negative real parts that represent decay, because of the strong dissipation $(-\partial^4 u/\partial x^4)$, on a time scale much shorter than that typical of the nonlinear terms.

We solve the thin film equation (5.32) in the time interval $0 \leq t \leq 1000$ with periodic boundary conditions and initial film thickness $\mathcal{H}(x, 0) = 1 + u(x, 0)$, where $u(x, 0) = 0$. We set $\alpha = 0.0048$ and $\gamma = 0.0532$ (taken from [36]) and use $N_{\mathcal{F}} = 32$ grid points in the Fourier spatial discretization. We utilize the ETD4RK method (5.13) with time-step size $\Delta t \approx 2^{-6}$ for the time-discretization. When evaluating the coefficients of the ETD4RK method (and similarly for the ETD and the ETD-RK methods utilized for the comparison computations presented in §5.5.1) via the 'Cauchy integral' approach [44, 45] (see §4.2.2), we choose circular contours of radius $R = 1$. Each contour is centered at one of the diagonal elements of the matrix of the linear part of the semi-discretized PDE (5.33). The contours are sampled at 32 equally spaced points and approximated by (4.16). The numerical results are presented in figure 5.16, which shows in (a) that the solution, at the polar angle $x = 0$, oscillates with time before eventually decaying to a steady state, shown in (b), at large time. In addition, the figure demonstrates excellent agreement between our numerical results and those of [36], providing a check on the accuracy of the method.

In §5.5.1, we solve numerically the thin film equation (5.32) up to final time $t = 20$ in a manner analogous to solving the former PDEs: the K-S (5.18) and the NLS (5.24) equations. We utilize the time-discretization methods listed in §5.2, and discuss the results of the comparison tests and state our conclusion in §5.5.2.

5.5.1 Computational Results

In our simulation tests, we numerically integrate the thin film equation (5.32) up to final times $t = 20$ with periodic boundary conditions and again the initial film thickness $\mathcal{H}(x, 0) = 1 + u(x, 0)$, where $u(x, 0) = 0$. We set $\alpha = 0.0048, \gamma = 0.0532$ (taken from [36]) and again use $N_{\mathcal{F}} = 32$ grid points in the Fourier spatial discretization.

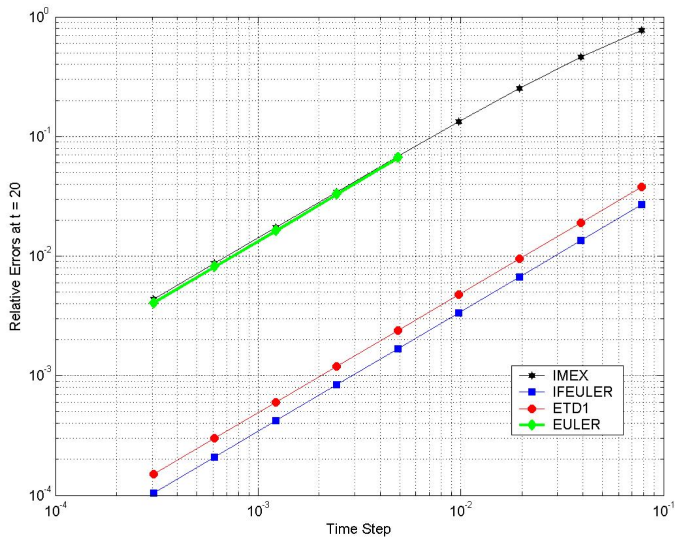
We measure the efficiency of each time-discretization method, listed in §5.2, in solving the test model, by computing the numerical relative error (5.17). The exact solution is approximated utilizing, for the time discretization, the ETD4RK method (5.13) with a very small time-step.

In figure 5.17, we plot in (a), (b) and (c) the accuracy of first, second and fourth-order methods as a function of the time step respectively. The aim is to look for the most competitive method which takes fewer steps (larger time-step size) to achieve a given error tolerance. Note that the accuracy is improved as time-step decreases, and that the figures confirm the order expected for each method. In figure 5.18 (a), (b) and (c), we plot the first, second and fourth-order methods' accuracy as a function of CPU time respectively, to add a competing factor in differentiating between the methods.

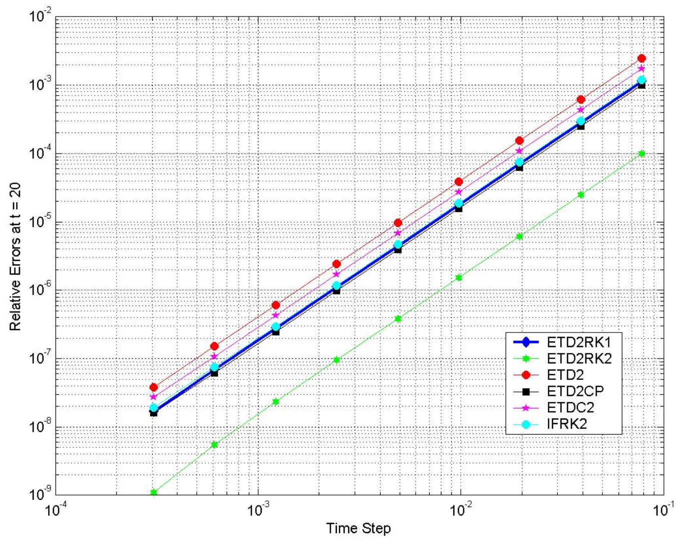
Considering the first-order methods, it appears from figure 5.17 (a) that the IMEX and the EULER methods are not reliable methods for solving the thin film equation (5.32). In the plot, we find that these two methods are the least accurate for a given time-step size. In addition, they are the most time consuming, see figure 5.18 (a), due to the very small time-step used by the methods (compared to the larger time-step used by the IFEULER and the ETD1 methods) to produce a solution that is accurate to any given level of accuracy. We find that the IFEULER method outperforms the EULER, the IMEX and the ETD1 methods in both accuracy and speed for any given level of accuracy.

To produce solutions for the thin film equation with higher orders of accuracy in time, we utilize second-order methods. In figure 5.17 (b), we find that, for a given time-step size, the IFRK2 method produces more accurate solutions than the IFEULER method does. However, for any given level of accuracy, it is the second most accurate and the most time consuming method, see figure 5.18 (b), comparing to other second-order methods. Its performance resembles that of the ETD2RK1 and the ETD2CP methods, though, the ETD2CP method is the second most costly

(a)



(b)



(c)

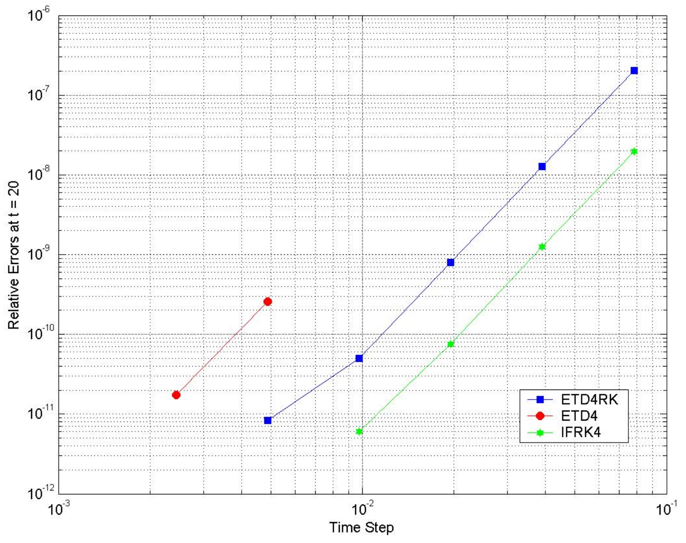
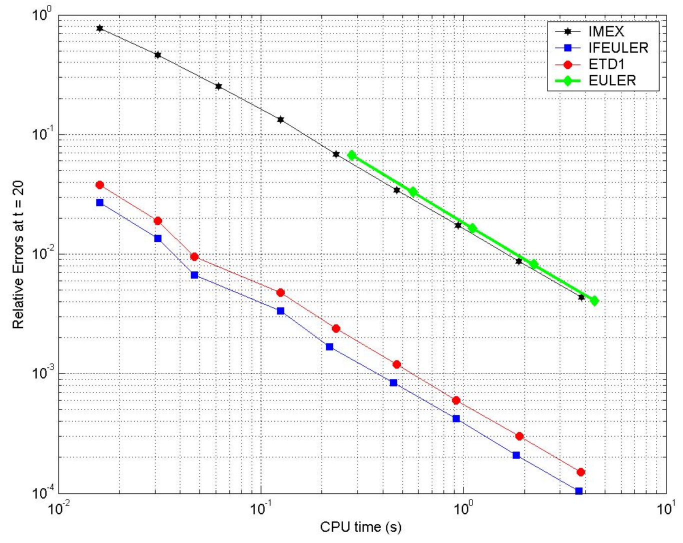
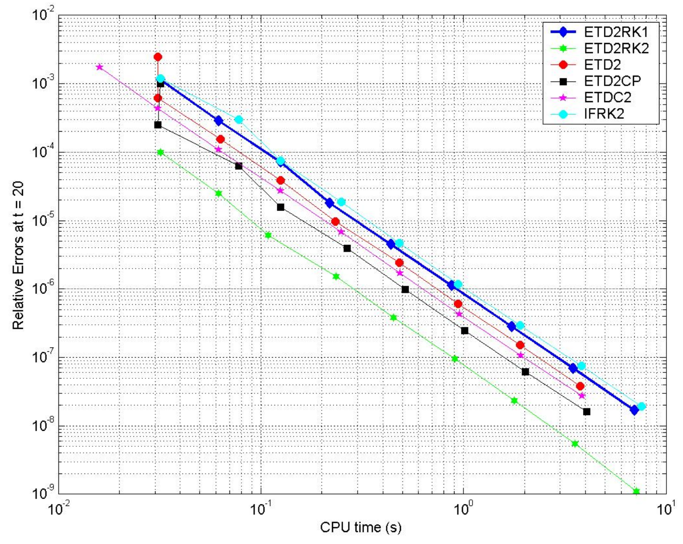


FIGURE 5.17: Relative errors versus time step for the thin film equation (5.32) with initial film thickness $\mathcal{H}(x,0) = 1$ (a) first-order methods (b) second-order methods (c) fourth-order methods.

(a)



(b)



(c)

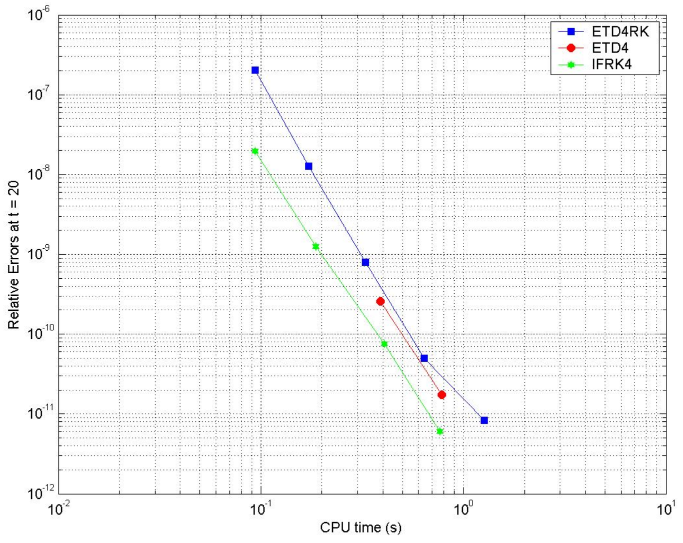


FIGURE 5.18: Relative errors versus CPU time for the thin film equation (5.32) with initial film thickness $\mathcal{H}(x,0) = 1$ (a) first-order methods (b) second-order methods (c) fourth-order methods.

method. In addition, tests show that the most accurate method is the ETD2RK2 method with the smallest amount of time consumed for solving the equation to any given level of accuracy, see figure 5.18 (b).

Finally, we consider utilizing fourth-order methods with fourth-order convergence, that guarantee higher order accuracy in time. As illustrated in figure 5.17 (c) and 5.18 (c), the IFRK4 method has proven to be a satisfactory method, being the most accurate and the least time consuming method for any given level of accuracy. In addition, our simulations reveal that, whereas the ETD4 fails to produce an accurate solution to the equation for time-step larger than $\Delta t \approx 5 \times 10^{-3}$, the ETD4RK method can use time-steps of a maximum size $\Delta t \approx 8 \times 10^{-2}$ to produce a solution with an accuracy of 10^{-7}, see figure 5.17 (c). This indicates that the ETD4RK method has larger stability region than that of the ETD4 method which in addition, is smaller than that of the ETD2 method. This agrees with our stability analysis in §3.3.

5.5.2 Conclusion

Problems in the fluid dynamics of thin films have been solved to demonstrate the effectiveness of exponential integrators. Under certain circumstances, we have found that, whereas the first-order IMEX, the EULER and the ETD4 methods are impractical for solving the nonlinear thin film equation (5.32), the IF and the ETD2RK2 methods have proven to be accurate and reliable. It would be interesting in future to analyze theoretically and understand the behavior of the numerical methods' performance in the experiments that have been conducted.

Our conclusions have relied on only one case study, where we have considered one fixed value of the surface-tension and the gravity parameters with an initially uniform film thickness. However, the thin film equation is strongly nonlinear, hence convergence and stability become solution-dependent issues, and our conclusions could differ greatly for different cases.

Chapter 6

Overall Conclusions

6.1 Overall Conclusions

This research aimed to employ **Exponential Time Differencing** (ETD) as a time-discretization method to solve accurately stiff partial differential equations. We considered the effectiveness of these methods for solving real application problems. Throughout this project, we also presented the modifications that these methods need in order to be effective. In essence, for semi-linear time-dependent equations, these schemes provide a systematic coupling of the explicit treatment of nonlinearities and the exact integration of the stiff linear part of the equations.

The thesis began with a review of the derivation of the explicit ETD method of arbitrary order s, which includes the explicit formula of the methods' coefficients, and we presented the Runge-Kutta (ETD-RK) methods of **Cox** and **Matthews** [19] up to fourth-order. We also derived the ETD2RK2 scheme (analogous to the "modified Euler" method [78]) as an example of the one-parameter family of the ETD2Rk$\jmath$ schemes for $\jmath \in \mathbb{R}^+$. We concluded that

- If the nonlinear part $F(u(t), t)$ of the differential equation (3.3) is zero, the ETD integrators produce the exact solution to the ODE and so the schemes are automatically A-stable.
- If the linear part is zero ($c = 0$ in (3.3)), the ETD and the ETD-RK integrators reduce to linear multi-step or classical explicit Runge-Kutta methods respectively.

This work raised the issue of defining other formulas for one-parameter families of s-order ETD-RK$\jmath$ schemes in future studies.

As a next step, we examined analytically the ETD and ETD-RK methods' stability properties, up to fourth-order. Tests were illustrated with figures where we computed and plotted the boundaries of the stability regions in two dimensions for negative and purely real values of the stiffness parameter in the test problem (3.24). The figures demonstrated that the stability regions of the ETD-RK methods are larger than those of the multi-step ETD methods, which agrees with the conventional fact that the RK methods have larger stability regions than the ordinary multi-step time-discretization methods of the same order. However, we found that the different types of an s-order ETD-RK schemes (for example, the ETD2RK1 and the ETD2RK2 schemes) have different stability regions, in contrast to the stability regions of different formulas of an s-order RK methods (which coincide). In addition, for any given value of the stiffness parameter, the stability regions of multi-step

ETD methods get smaller as the order of the methods increases, which agrees with the stability characteristic of the ordinary multi-step methods. This work illustrates that the ETD and the ETD-RK methods have the advantage of avoiding the severe restrictions on the time-step size when compared with any conventional explicit method in solving a stiff system of ODEs. We found that the stability regions of the ETD and ETD-RK methods grow larger as the stiffness parameter decreases, which permits the usage of a large time-step size and consequent rapidity in computations.

Applying the ETD methods requires the computation of the coefficients, which are matrix exponentials and related matrix functions of the linear operators. A complication [19] arises in the computation of these coefficients, in addition to the difficulties already inherent in computing a matrix exponential [60]. For matrices which have eigenvalues equal to zero, the explicit formulas of the coefficients involve division by zero, while for matrices which have very small eigenvalues approaching zero, the coefficients suffer from rounding errors due to the large amount of cancellation in the formulas.

At this stage of the research, the plan was to test various algorithms against each other and assess their accuracy, efficiency and ease of use for improving the numerical difficulties in approximating the ETD coefficients and for an efficient implementation of the ETD methods. The algorithms studied in this thesis are the Taylor series, the Cauchy integral formula, the Scaling and Squaring algorithm, the Composite Matrix algorithm and the Matrix Decomposition algorithm for non-diagonal matrix cases.

We now reiterate the main conclusions that were drawn from this work:

1. **Taylor Series:** This algorithm is known for its simplicity and ease of implementation. However, its efficiency deteriorates when approximating the ETD coefficients for large values (in magnitude) of the argument (matrix norm in the matrix case).

2. **The Cauchy Integral Formula:** The algorithm was proposed by **Kassam** and **Trefethen** [44, 45] to evaluate the ETD coefficients by means of contour integration in the complex plane approximated by the Trapezium rule (4.16).

   This algorithm turned out to be very accurate for diagonal matrix problems, but it can be inaccurate and time consuming for non-diagonal matrices with large norm. In addition, this algorithm requires a prior knowledge of the eigenvalue of largest magnitude, and we must ensure that none of the points on the contour are close to or at the origin, otherwise the original problem of rounding errors reappears.

We gave theoretical estimate formulas of the errors when using the Cauchy integral formula to approximate the ETD coefficients for matrices with large norm, and we used these formulas to estimate the number of points required to discretize the contour to achieve a relative error of some chosen tolerance. However, improvements to this algorithm have recently been developed [70].

3. **Scaling and Squaring Algorithm Type I:** This algorithm is one of the most effective and powerful algorithms for diagonal and non-diagonal matrix problems. However, it is the most complex to implement.

   In diagonal matrix cases, it is stable for small positive values and for all negative values on the diagonal. However, the algorithm performance deteriorates for large positive values. Due to our analysis, we found that the errors resulting from the scaling and squaring process in approximating, for large positive values, either the exponential of a diagonal matrix (when the algorithm's formulas include it) or the identity (4.27) (if the algorithm is based on it), are doubled at each scaling. The analysis in both cases also predicts that these errors increase linearly as the positive values increase.

   In non-diagonal matrix cases, the algorithm requires the knowledge of the eigenvalue of largest magnitude. In the case of matrices with negative eigenvalues, we found that the performance of the algorithm, when it is based on the identities (4.27), (4.21) and (4.22), agrees well with that in case of the negative values of the diagonal matrix. This is due to the advantage that we do not need to compute a matrix exponential. However, when the algorithm is based on (4.20) - (4.22), or (4.23) - (4.25) or (4.27) and (4.24) - (4.25), we found that the errors resulting from the scaling and squaring process cause the performance of the algorithm to deteriorate for large norm matrices. This behavior is contrary to that in the case of negative values of the diagonal matrices, and it is consequently a task for future research to investigate.

   Note that we favored Taylor series to combine with the algorithm rather than the popular Padé approximation. Firstly, we found that Padé approximations lead to larger rounding errors, due to cancellation problems, than

Taylor series, which are then amplified by the scaling and squaring process. Secondly, Padé approximations require a more expensive matrix inversion, in which the matrix can be very poorly conditioned with respect to inversion.

4. **Scaling and Squaring Algorithm Type II:** This algorithm gave accurate results when evaluating the coefficients in the first-order ETD1 method (3.14) for very small values (in magnitude) of the argument utilizing the identity (4.48). Due to our analysis, we found that there is no amplification of the errors and the algorithm's accuracy remains the same at each scaling. However, this algorithm did not perform well when computing the coefficients in higher order ETD methods for very small values (in magnitude) of the argument, due to the amplification of rounding errors at each scaling as our analysis suggested. Thus, the Scaling and Squaring type **II** algorithm is not generally useful.

5. **Composite Matrix Algorithm:** Implementing this algorithm involves taking the exponential of a specially constructed matrix, via the Matlab routine "*expm*", which is based on the Scaling and Squaring algorithm. The resulting matrix contains the values of the ETD coefficients which then can be extracted easily.

   For small positive values and all negative values in diagonal problems and for small norm matrices in non-diagonal problems, the algorithm proved to be successful in approximating the ETD coefficients accurately. But it is inaccurate and computationally expensive in time for non-diagonal matrices with large norm, due to the larger number of scaling and squaring process that become inaccurate.

6. **Matrix Decomposition Algorithm:** This algorithm simplifies the evaluation of a function of a non-diagonal matrix exponential to that of a diagonal matrix exponential whose elements are the eigenvalues of the non-diagonal matrix. This algorithm is remarkably accurate when compared with the explicit formula for ETD coefficients, and is the cheapest algorithm in time. For small norm matrices, however, it is slightly less accurate than the Cauchy integral formula, the Scaling and Squaring type **I** and the Composite Matrix algorithms.

Tests on the second-order centered difference differentiation matrix for the first and second derivatives and the Chebyshev differentiation matrix for the second derivative exhibit qualitatively similar results, except that the errors are typically larger for the Chebyshev matrix, due to the larger eigenvalues of this matrix.

The above results led us to agree with the quotation "practical implementations are dubious in the sense that implementation of a sole algorithm might not be entirely reliable for all classes of problems" [60]. However, in differentiating between the algorithms considered, we concluded that the Scaling and Squaring type **I** algorithm is an efficient algorithm for computing the ETD coefficients in both diagonal and non-diagonal matrix cases. It exhibits some loss of accuracy for large values of the scalar arguments and large norm of matrices, but this is much less severe than for the Taylor series and the Cauchy integral formula. Also, it compares favorably with the high computational cost of the Cauchy integral formula and the Composite Matrix algorithm in non-diagonal matrix cases. The Matrix Decomposition algorithm, in the conventional eigenvector approach is also very efficient computationally, though it is slightly less accurate when the matrix norm is small, and is not applicable to matrices that do not have a complete set of linearly independent eigenvectors (where no invertible matrix of eigenvectors exists).

The final part of this project aimed to conduct numerical comparison experiments on three stiff PDEs, in one space dimension. We employed first, second and fourth-order ETD methods, including the ETD and the ETD-RK methods proposed by **Cox** and **Matthews** [19], and made some observations regarding their efficiency against other competing stiff integrators including: the first-order Implicit-Explicit (IMEX) method and first, second and fourth-order Integrating Factor (IF) methods. The problems considered were: the dissipative time-dependent scalar **Kuramoto-Sivashinsky (K-S)** equation, the nonlinear dispersive **Schrödinger (NLS)** equation and the nonlinear (dissipative-dispersive) **Thin Film** equation. In the K-S and the NLS equations, the linear terms of the equations are primarily responsible for stiffness whereas in the thin film equation the nonlinear terms are the stiffest.

For the simulation tests, we chose periodic boundary conditions and applied Fourier spectral approximation for the spatial discretization. In addition, we evaluated the coefficients of the ETD and the ETD-RK methods via the 'Cauchy integral' approach [44, 45].

Our simulations revealed considerably different performances of the compared numerical methods in different cases. Regarding accuracy and CPU time in the solving process of the K-S equation (5.20), we concluded that the ETD4RK method (5.13) is marginally the best. It maintains good stability and produces high accuracy with reasonable computational effort. Furthermore, when solving the test model for the specific initial condition (5.22), we found that the ETD and ETD-RK methods of [19] outperformed the compared methods in both speed and accuracy.

For non-traveling or slowly traveling wave soliton solutions of the NLS equation (5.24), we found that, the most efficient methods of those we compared are the ETD and ETD-RK methods of [19]. However, the performance of these methods declines for solutions with larger speeds (fast traveling waves) and the IF methods then prove to be the most accurate. Our analysis revealed that, as the soliton wave-speed increases, the local truncation error of the ETD methods gets larger and the methods become less accurate. On the other hand, the local truncation error of the IF methods does not change as the speed varies and hence these methods maintain their performance, and moreover we obtain the same quantitative results for the errors (over the same range of time-step sizes) for an s-order IF method.

To apply our numerical tests to the nonlinear thin film equation, we introduced a perturbation to split the equation into linear and nonlinear terms. For this equation we found that the first-order IMEX and the ETD4 methods are impractical methods, whereas the IF and the ETD2RK2 methods proved to be accurate and reliable. It would also be interesting to analyze theoretically and understand in future work the behavior of the numerical methods' performance. Further studies on the thin film equation should consider large and small perturbations to the constant solution. We expect exponential integrators to perform well when the perturbation is small. Large perturbations lead to nonlinear terms with a stiffer character, and hence the performance of the exponential integrators could deteriorate.

In addition, we should consider cases of varying the surface-tension and the gravity parameters in equation (5.32). For example, increasing the surface-tension makes the decay of the amplitude of the higher oscillating Fourier modes more rapid and the complexity of the time-dependent solutions increases rapidly.

Overall we deduced that all the compared methods exhibited the order of accuracy expected, and proved to be efficient alternatives to standard explicit integrators for computing solutions for stiff problems without severe time-step size restrictions. Additionally, we noted that, for accurate and economical computations, it is often advantageous to utilize fourth-order methods. The benefits of these methods are that they can use a much larger time-step size than the lower order comparable integrators for an equivalent level of accuracy, and hence they are cheap. We also found that the ETD integrators rely on the fast evaluation of the exponential and related functions. The computations of the methods' coefficients, which are done once at the beginning of the integration for each time step size, have a noticeable effect on the CPU times, as they impose a significant timing overhead when the methods use a large time step. However, ETD schemes can be efficiently combined with spatial approximations to provide accurate smooth solutions for stiff or highly oscillatory semi-linear PDEs. These methods were shown to perform extremely well in solving various real application problems, while achieving high accuracy and maintaining good stability. The ETD-RK methods were demonstrated to be more stable and allow to use larger time-step size than that used by the multi-step ETD methods, and we also found that the lower order multi-step ETD methods are more stable than higher order ones, which agrees very well with our stability analysis in §3.

As a final point, we caution that our conclusions are restricted only to the cases studied. These results cannot be generalized, as they may differ for other choices of initial conditions and for other problems. It is clear that the best choice of method depends on the specific problem to be solved.

Our research serves as a basis for more detailed theoretical and numerical investigations on time-discretization methods to be carried out in the future. It is hoped that the future investigation will serve dual roles. Firstly, to confirm that the ETD methods can be ideal methods to cope with stiff systems in a wide range of applications. Secondly, to develop time-discretization methods that can facilitate numerical studies of higher-order problems with nonlinear stiff terms arising from mathematical models of a diverse range of physical phenomena.

Appendices

Appendix A

The Numerical Solution of the Kuramoto-Sivashinsky Equation

This **Matlab** program is used to obtain the numerical solution of the **Kuramoto-Sivashinsky (K-S)** equation (5.18), utilizing the ETD4RK method (5.13), and to produce figure 5.2.

```
% Spatial grid
N = 64; L = 2*pi; x = ([0:N-1]*2*L/N)'; dt = 2^(-10);

% Spectral differentiation matrices
D1 = i*(pi/L)*[0:N/2-1 0 -N/2+1:-1];
D2 = D1.^2; D2((N/2)+1) = -(N*pi/(2*L))^2;
D4 = D2.^2; c = -D2-D4;

% Evaluating the coefficients of the ETD4RK method
% Using Cauchy integral formula
R = 1; N1 = 32; r = R*exp(2*i*pi*(1:N1)/N1);
c1 = c*dt; c2 = c1/2; E1 = exp(c1); E = exp(c2);

for k = 1:N
  C1(k) = real(mean((dt/2)*((exp(c2(k)+r)-1)./(c2(k)+r))));
  C2(k) = real(mean(dt*((-4-c1(k)-r+exp(c1(k)+r).*(4-3*(c1(k)+r)+(c1(k)+r).^2))
         ./(c1(k)+r).^3)));
  C3(k) = real(mean(dt*((2+c1(k)+r+(c1(k)+r-2).*exp(c1(k)+r))./(c1(k)+r).^3)));
  C4(k) = real(mean(dt*((-4-3*(c1(k)+r)-(c1(k)+r).^2+(4-c1(k)-r).*exp(c1(k)+r))
         ./(c1(k)+r).^3)));
end

% Initial condition
u = exp(cos(x/2)); uhat = fft(u);

% Solve PDE:
tmax = 60; nmax = round(tmax/dt); nc = 60; nplt = floor(nmax/nc);
udata = u; tdata = 0;
min1 = min(u); max1 = max(u);

for n = 1:nmax
  t = n*dt;
  uhat1_x = D1.*fft(u.^2)/2;
  ahat = (E.*uhat)-(C1'.*uhat1_x); a=real(ifft(ahat));
  bhat = (E.*uhat)-(C1'.*D1.*fft(a.^2)/2); b=real(ifft(bhat));
  chat = (E.*ahat)-(C1'.*(D1.*fft(b.^2)-uhat1_x)); C=real(ifft(chat));
  uhat = (E1.*uhat)-(C2'.*uhat1_x+C3'.*D1.*(fft(a.^2)+fft(b.^2))
         +C4'.*D1.*fft(C.^2)/2);
  u = real(ifft(uhat));
 if mod(n,nplt) == 0
    udata = [udata u]; tdata = [tdata t];
    min1 = [min1 min(u)]; max1 = [max1 max(u)];
  end
end

% plot results:
set(gcf,'renderer','zbuffer'), clf, drawnow
mesh(x,tdata,udata'), colormap(1e-6*[1 1 1])
xlabel x, ylabel t, zlabel u, grid on
axis([0 2*L 0 tmax floor(min(min1)) ceil(max(max1))])
set(gca,'ztick',[floor(min(min1)) ceil(max(max1))])
```

Appendix B

Derivation of the Local Truncation Errors

Local truncation errors or discretization errors are errors made by numerical algorithms that arises from taking finite number of steps in computation. It is present even with infinite-precision arithmetic, because it is caused by truncation of the infinite Taylor series to form the algorithm.

To derive the local truncation errors $L.T.E_1$ (5.28a)

$$L.T.E_1 \approx \frac{\Delta t^2}{2} dF(u(t),t)/dt,$$

of the ETD1 method (5.2)

$$u(t_{n+1}) = u(t_n)e^{c\Delta t} + (e^{c\Delta t} - 1)F(u(t_n), t_n)/c,$$

and $L.T.E_2$ (5.28b)

$$L.T.E_2 \approx \frac{\Delta t^2}{2} d(F(u(t),t)e^{-ct})/dt,$$

of the IFEULER method (5.4)

$$u(t_{n+1}) = (u(t_n) + \Delta t F(u(t_n), t_n))e^{c\Delta t},$$

(both methods are performed with respect to the model $du(t)/dt = cu(t) + F(u(t),t)$ (5.1)) let us assume that the function $u(t_{n+1})$ can be expanded formally in Taylor series about t_n as follows,

$$u(t_{n+1}) = u(t_n) + \Delta t \left.\frac{du(t)}{dt}\right|_{t=t_n} + \frac{\Delta t^2}{2!} \left.\frac{d^2u(t)}{dt^2}\right|_{t=t_n} + \frac{\Delta t^3}{3!} \left.\frac{d^3u(t)}{dt^3}\right|_{t=t_n} + \cdots , \tag{B.1}$$

where

$$\begin{aligned}
\frac{d^2u(t)}{dt^2} &= c\frac{du(t)}{dt} + \frac{dF(u(t),t)}{dt}\\
&= c^2u(t) + cF(u(t),t) + \frac{dF(u(t),t)}{dt}, \qquad \text{(B.2a)}\\
\frac{d^3u(t)}{dt^3} &= c\frac{d^2u(t)}{dt^2} + \frac{d^2F(u(t),t)}{dt^2},\\
&= c^3u(t) + c^2F(u(t),t) + c\frac{dF(u(t),t)}{dt} + \frac{d^2F(u(t),t)}{dt^2}, \qquad \text{(B.2b)}\\
&\vdots\\
\frac{d^mu(t)}{dt^m} &= c\frac{d^{m-1}u(t)}{dt^{m-1}} + \frac{d^{m-1}F(u(t),t)}{dt^{m-1}},\\
&= c^mu(t) + c^{m-1}F(u(t),t) + c^{m-2}\frac{dF(u(t),t)}{dt} + \cdots\\
&+ c\frac{d^{m-2}F(u(t),t)}{dt^{m-2}} + \frac{d^{m-1}F(u(t),t)}{dt^{m-1}}. \qquad \text{(B.2c)}
\end{aligned}$$

For the ETD1 method (5.2), expand $u(t_{n+1})$ utilizing (B.1), and substitute the Taylor series expansion of the exponential function $e^{c\Delta t}$ to deduce

$$u(t_n) + \Delta t\left.\frac{du(t)}{dt}\right|_{t=t_n} + \frac{\Delta t^2}{2!}\left.\frac{d^2u(t)}{dt^2}\right|_{t=t_n} + \cdots = \Big(1 + c\Delta t + \frac{(c\Delta t)^2}{2!} + \cdots\Big)u(t_n) + \Big(\Delta t + \frac{c\Delta t^2}{2!} + \cdots\Big)F(u(t_n),t_n). \qquad \text{(B.3)}$$

Subtracting equivalent terms and substituting (B.2) in (B.3) gives us

$$\Delta t(cu(t_n) + F(u(t_n),t_n)) + \frac{\Delta t^2}{2!}\Big(c^2u(t_n) + cF(u(t_n),t_n) + \left.\frac{dF(u(t),t)}{dt}\right|_{t=t_n}\Big) + \cdots = \Big(c\Delta t + \frac{(c\Delta t)^2}{2!} + \cdots\Big)u(t_n) + \Big(\Delta t + \frac{c\Delta t^2}{2!} + \cdots\Big)F(u(t_n),t_n).$$

Again, subtract equivalent terms in the above equation to deduce the local truncation error (5.28a) of the ETD1 method

$$L.T.E_1 \approx \frac{\Delta t^2}{2}dF(u(t),t)/dt.$$

For the IFEULER method (5.4)

$$u(t_{n+1})e^{-ct_{n+1}} = (u(t_n) + \Delta tF(u(t_n),t_n))e^{-ct_n},$$

expand $u(t_{n+1})e^{-ct_{n+1}}$ utilizing (B.1)

$$u(t_n)e^{-ct_n} + \Delta t\left.\frac{d(u(t)e^{-ct})}{dt}\right|_{t=t_n} + \frac{\Delta t^2}{2!}\left.\frac{d^2(u(t)e^{-ct})}{dt^2}\right|_{t=t_n} + \cdots = (u(t_n) + \Delta tF(u(t_n),t_n))e^{-ct_n},$$

then subtract equivalent terms to deduce

$$\Big(\Delta t\Big(\left.\frac{du(t)}{dt}\right|_{t=t_n} - cu(t_n) - F(u(t_n),t_n)\Big) + \frac{\Delta t^2}{2!}\Big(\left.\frac{d^2u(t)}{dt^2}\right|_{t=t_n} - 2c\left.\frac{du(t)}{dt}\right|_{t=t_n} + c^2u(t_n)\Big) + \cdots\Big)e^{-ct_n} = 0. \qquad \text{(B.4)}$$

Substituting (B.2) in (B.4) gives us

$$\frac{\Delta t^2}{2!}\Big(\frac{dF(u(t),t)}{dt}\Big|_{t=t_n} - cF(u(t_n),t_n)\Big)e^{-ct_n} + \cdots = 0.$$

Thus, the local truncation error (5.28b) of the IFEULER method is

$$L.T.E_2 \approx \frac{\Delta t^2}{2} d(F(u(t),t)e^{-ct})/dt.$$

Bibliography

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions.* Dover Publications Inc., 1972.

[2] F. Aluffi-Pentini, V. DeFonzo, and V. Parisi. A Novel Algorithm for the Numerical Integration of Systems of Ordinary Differential Equation Arising in Chemical Problems. *Journal of Mathematical Chemistry*, 33:1–15, 2003.

[3] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri. Implicit-Explicit Runge-Kutta Methods for Time-Dependent Partial Differential Equations. *Appl. Num. Math.*, 25:151–167, 1997.

[4] U. M. Ascher, S. J. Ruuth, and B. T.R. Wetton. Implicit-Explicit Methods for Time-Dependent Partial Differential Equations. *SIAM J. Numer. Anal.*, 32:797–823, 1995.

[5] H.A. Ashi, L.J. Cummings, and P.C. Matthews. Comparison of Methods for Evaluating Functions of a Matrix Exponential. *Applied Numerical Mathematics*, 59:468–486, 2009.

[6] L. R. Band, D. S. Riley, P. C. Matthews, J. M. Oliver, and S. L. Waters. Annular Thin-Film Flows Driven by Azimuthal Variations in Inter-Facial Tension. *Quart. J. Mech. Appl. Math.*, In press.

[7] H. Berland and B. Skaflestad. Solving the Nonlinear Schrödinger Equation Using Exponential Integrators. *Norwegian Society of Automatic Control*, 27:201–217, 2006.

[8] H. Berland, B. Skaflestad, and W. M. Wright. EXPINT - A Matlab Package for Exponential Integrators. *ACM Transactions on Mathematical Software*, 33:Article Number 4, 2007.

[9] G. Beylkin, J. M. Keiser, and L. Vozovoi. A New Class of Time Discretization Schemes for the Solution of Nonlinear PDEs. *J. Comput. Phys.*, 147:362–387, 1998.

[10] J. Billingham and A. C. King. *Wave Motion.* Cambridge University Press, 2000.

[11] J. P. Boyd. *Chebyshev and Fourier Spectral Methods.* Dover, New York, second edition, 2001.

[12] E. O. Brigham. *The Fast Fourier Transform and its Applications.* Prentice-Hall, Englewood Cliffs, NJ, 1988.

[13] J. C. Bronski and J. N. Kutz. Numerical Simulation of the Semi-Classical Limit of the Focusing Nonlinear Schrödinger Equation. *Phy. Lett. A*, 245:325–336, 1999.

[14] R. L. Burden and J. D. Faires. *Numerical Analysis.* Wadsworth Group, seventh edition, 2001.

[15] M. Calvo and C. Palencia. A Class of Explicit Multi-Step Exponential Integrators for Semi-Linear Problems. *Numer. Math.*, 102:367–381, 2006.

[16] M. P. Calvo, J. de Frutos, and J. Novo. Linearly Implicit Runge-Kutta Methods for Advection-Reaction-Diffusion Equations. *Appl. Num. Math.*, 37:535–549, 2001.

[17] J. Certaine. The Solution of Ordinary Differential Equations with Large Time Constants. *In Mathematical Methods for Digital Computers*, A. Ralston and H. S. Wilf, eds.:128–132, Wiley, New York, 1960.

[18] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comp.*, 19:297–301, 1965.

[19] S. M. Cox and P. C. Matthews. Exponential Time Differencing for Stiff Systems. *J. Comput. Phys.*, 176:430–455, 2002.

[20] C. F. Curtiss and J. O. Hirschfelder. Integration of Stiff Equations. *Proc. Nat. Acad. Sci.*, 38:235–243, 1952.

[21] G. Dahlquist. A Special Stability Problem for Linear Multi-Step. *BIT Numer. Math.*, 3:27–43, 1963.

[22] Q. Du and W. Zhu. Stability Analysis and Applications of the Exponential Time Differencing Schemes. *Journal of Computational Mathematics*, 22:200–209, 2004.

[23] Q. Du and W. Zhu. Analysis and Applications of the Exponential Time Differencing Schemes and their Contour Integration Modifications. *BIT Numer. Math.*, 45:307–328, 2005.

[24] P. L. Evans, L. W. Schwartz, and R. V. Roy. Steady and Unsteady Solutions for Coating Flow on a Rotating Horizontal Cylinder: Two-Dimensional Theoretical and Numerical Modeling. *Phys. Fluids*, 16:2742–2756, 2004.

[25] B. Fornberg. *A Practical Guide to Pseudo-Spectral Methods.* Cambridge University Press, Cambridge, UK, 1996.

[26] B. Fornberg and T. A. Driscoll. A Fast Spectral Algorithm for Nonlinear Wave Equations with Linear Dispersion. *J. Comput. Phys.*, 155:456–467, 1999.

[27] J. Frank, W. Hundsdorfer, and J. G. Verwer. On the Stability of Implicit-Explicit Linear Multi-Step Methods. *Appl. Num. Math.*, 25:193–205, 1997.

[28] A. Friedli. Generalized Runge-Kutta Methods for the Solution of Stiff Differential Equations. *In Numerical Treatment of Differential Equations, R. Burlirsch, R. Grigorieff, and J. Schröder, eds., 631*, Lecture Notes in Mathematics:35–50, Springer, Berlin, 1978.

[29] U. Frisch, Z. S. She, and O. Thual. Viscoelastic Behavior of Cellular Solution to the Kuramoto-Sivashinsky Model. *J. Fluid. Mech.*, 168:221–240, 1986.

[30] D. Garfinkel, C. B. Marbach, and N. Z. Shapiro. Stiff Differential Equations. *Ann. Rev. Biophys*, 6:525–542, 1977.

[31] C. W. Gear. Automatic Integration of Ordinary Differential Equations. *Communications of the ACM*, 14:176–179, 1971.

[32] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[33] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations* II. Springer-Verlag, Berlin, second edition, 1996.

[34] P. Henrici. Fast Fourier Methods in Computational Complex Analysis. *SIAM Review*, 21:481–527, 1979.

[35] N. J. Higham. The Scaling and Squaring Method for the Matrix Exponentials Revisited. *SIAM J. Matrix Anal. Appl.*, 26:1179–1193, 2005.

[36] E. J. Hinch and M. A. Kelmanson. On the Decay and Drift of Free-Surface Perturbations in Viscous Thin-Film Flow Exterior to a Rotating Cylinder. *Proc. R. Soc. Lond. A*, 459:1193–1213, 2003.

[37] M. Hochbruck, C. Lubich, and H. Selhofer. Exponential Integrators for Large Systems of Differential Equations. *SIAM. J. Sci. Comp.*, 19:1552–1574, 1998.

[38] M. Hochbruck and A. Ostermann. Explicit Exponential Runge-Kutta Methods for Semi-linear Parabolic Problems. *SIAM J. Numer. Anal.*, 43:1069–1090, 2005.

[39] M. Hochbruck and A. Ostermann. Exponential Runge-Kutta Methods for Parabolic Problems. *Appl. Numer. Math.*, 53:323–339, 2005.

[40] R. Holland. Finite-Difference Time-Domain (FDTD) Analysis of Magnetic Diffusion. *IEEE Trans. Electromagn. Compat.*, 36:32–39, 1994.

[41] J. M. Hyman and B. Nicolanenko. The Kuramoto-Sivashinsky Equation: a Bridge Between PDE's and Dynamical Systems. *Physica D.*, 18:113–126, 1986.

[42] E. Infeld and G. Rowlands. *Nonlinear Waves, Solitons and Chaos.* Cambridge University Press, 1990.

[43] A. Iserles. *A First Course in the Numerical Analysis of Differential Equations.* Cambridge University Press, Cambridge, UK, 1996.

[44] A. K. Kassam. *High Order Time stepping for Stiff Semi-Linear Partial Differential Equations.* PhD thesis, Oxford University, 2004.

[45] A. K. Kassam and L. N. Trefethen. Fourth-Order Time Stepping for Stiff PDEs. *SIAM J. Sci. Comput.*, 26:1214–1233, 2005.

[46] C. Klein. Fourth Order Time-Stepping for Low Dispersion Korteweg-de Vries and Nonlinear Schrödinger Equations. *Electronic Transactions on Numerical Analysis*, 29:116–135, 2008.

[47] S. Koikari. An Error Analysis of the Modified Scaling and Squaring Method. *Comput. Math. Appl.*, 53:1293–1305, 2007.

[48] H.-O. Kreiss and J. Oliger. Comparison of Accurate Methods for the Integration of Hyperbolic Equations. *Tellus*, 24:199–215, 1972.

[49] S. Krogstad. Generalized Integrating Factor Methods for Stiff PDEs. *J. Comput. Phys.*, 203:72–88, 2005.

[50] Y. Kuramoto. Diffusion-Induced Chaos in Reaction Systems. *Progr. Theoret. Phys. Suppl.*, 64:346–367, 1978.

[51] C. Lanczos. Trigonometric Interpolation of Empirical and Analytical Functions. *J. Math. Phys.*, 17:123–199, 1938.

[52] J. D. Lawson. Generalized Runge-Kutta Processes for Stable Systems with Large Lipschitz Constants. *SIAM J. Numer Anal.*, 4:372–380, 1967.

[53] P. W. Livermore. An Implementation of the Exponential Time Differencing Scheme to the Magnetohydrodynamics Equations in a Spherical Shell. *J. Comput. Phys.*, 220:824–838, 2007.

[54] Y. Y. Lu. Computing a Matrix Function for Exponential Integrators. *J. Comput. Appl. Math.*, 161:203–216, 2003.

[55] J. E. Marsden and M. J. Hoffman. *Basic Complex Analysis.* W. H. Freeman and Company, third edition, 1998.

[56] B. V. Minchev. Computing Analytic Matrix Functions for a Class of Exponential Integrators. Reports in Informatics 278, University of Bergen, Bergen, Norway, 2004.

[57] B. V. Minchev and W. M. Wright. A Review of Exponential Integrators for First Order Semi-Linear Problems. *Tech. Rep. NTNU. (2005)*, , Preprint.

[58] A. R. Mitchell and D. F. Griffiths. *The Finite Difference Method in Partial Differential Equations*. John Wiley & Sons, 1980.

[59] C. Moler and C. Van Loan. Nineteen Dubious Ways to Compute the Exponential of a Matrix. *SIAM Review*, 20:801–836, 1978.

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

[61] D. R. Mott, E. S. Oran, and B. V. Leer. A Quasi-Steady-State Solver for the Stiff Ordinary Differential Equations of Reaction Kinetics. *J. Comput. Phys.*, 164:407–428, 2000.

[62] J. D. Murray. *Mathematical Biology*. Springer-Verlag Berlin Heidelberg, second edition, 1993.

[63] S. P. Nørsett. An A-Stable Modification of the Adams-Bashforth Methods. *In Conf. on Numerical Solution of Differential Equations*, Lecture Notes in Math. 109/1969:214–219, Springer-Verlag, Berlin, 1969.

[64] M.S. Paterson and L. J. Stockmeyer. On the Number of Non-Scalar Multiplications Necessary to Evaluate Polynomials. *SIAM J. Comput.*, 2:60–66, 1973.

[65] P. G. Petropoulos. Analysis of Exponential Time-Differencing for FDTD in Lossy Dielectrics. *IEEE Trans. on Antennas and Propagation*, 45:1054–1057, 1997.

[66] S. J. Ruuth. Implicit-Explicit Methods for Reaction-Diffusion Problems in Pattern Formation. *J. Math. Biol.*, 34:148–176, 1995.

[67] Y. Saad. Analysis of Some Krylov Subspace Approximations to the Matrix Exponential Operator. *SIAM J. Numer. Anal.*, 29:209–228, 1992.

[68] J. M. Sanz-Serna and J. G. Verwer. Special Issue on Time Integration. *Applied Numerical Mathematics*, 25:135–136, 1997.

[69] T. Schmelzer. *The Fast Evaluation of Matrix Functions for Exponential Integrators*. PhD thesis, Oxford University, 2007.

[70] T. Schmelzer and L. N. Trefethen. Evaluating Matrix Functions for Exponential Integrators via Carathéodory–Fejér Approximation and Contour Integrals. *Electronic Transactions on Numerical Analysis*, 29:1–18, 2007.

[71] C. Schuster, A. Christ, and Fichtner W. Review of FDTD Time-Stepping for Efficient Simulation of Electric Conductive Media. *Microwave Optical Technol. Lett.*, 25:16–21, 2000.

[72] L. F. Shampine and C. W. Gear. A User's View of Solving Stiff Ordinary Differential Equations. *SIAM Review*, 21:1–17, 1979.

[73] R. B. Sidje. EXPOKIT: A Software Package for Computing Matrix Exponentials. *ACM Trans. Math. Softw.*, 24:130–156, 1998.

[74] G. I. Sivashinsky. Nonlinear Analysis of Hydrodynamic Instability in Laminar Flames, Part I: Derivation of the Basic Equations. *Acta Astronautica*, 4:1176–1206, 1977.

[75] G. I. Sivashinsky. Instabilities, Pattern Formation, and Turbulence in Flames. *Ann. Rev. Fluid Mech.*, 15:179–199, 1983.

[76] B. Skaflestad and W. M. Wright. The Scaling and Modified Squaring Method for Matrix Functions Related to the Exponential. *Applied Numerical Mathematics*, 59:783–799, 2009.

[77] C. Sulem and P. Sulem. *The Nonlinear Schrödinger Equation: Self-Focusing and Wave Collapse.* Springer-Verlag New York, 1999.

[78] E. Süli and D. F. Mayers. *An Introduction to Numerical Analysis.* Cambridge University Press, first edition, 2003.

[79] E. Tadmor. The Exponential Accuracy of Fourier and Chebyshev Differencing Methods. *SIAM J. Numer. Anal.*, 23:1–10, 1986.

[80] H. Tal-Ezer. Spectral Methods in Time for Parabolic Problems. *SIAM J. Numer. Anal.*, 26:1–11, 1989.

[81] M. Tokman. Efficient Integration of Large Stiff Systems of ODEs with Exponential Propagation Iterative (EPI) Methods. *J. Comput. Phys.*, 213:748–776, 2006.

[82] C. E. Treanor. A Method for the Numerical Integration of Coupled First-Order Differential Equations with Greatly Different Time Constant. *Math. Comp.*, 20:39–45, 1966.

[83] L. N. Trefethen. *Finite Difference and Spectral Methods for Ordinary and Partial Differential Equations.* Online at: http://www.comlab.ox.ac.uk/nick.trefethen/pdetext.html.

[84] L. N. Trefethen. *Spectral Methods in MATLAB.* SIAM, Philadelphia, 2000.

[85] L. N. Trefethen and H. M. Gutknecht. The Carathéodory-Fejér Method for Real Rational Approximation. *SIAM J. Numer. Anal.*, 20:420–436, 1983.

[86] C. F. Van Loan. A Note on the Evaluation of Matrix Polynomials. *IEEE Trans. Automatic Control*, AC-24:320–321, 1979.

[87] J. M. Varah. Stability Restrictions on Second Order, Three Level Finite Difference Schemes for Parabolic Equations. *SIAM J. Numer. Anal.*, 17:300–309, 1980.

[88] T. P. Witelski and M. Bowen. ADI Schemes for Higher-Order Nonlinear Diffusion Equations. *Appl. Num. Math.*, 45:331–351, 2003.

[89] W. Wright. *A Partial History of Exponential Integrators.* Department of Mathematical Sciences, NTNU, Norway, Online at: http://www.math.ntnu.no/num/expint/talks/wright04innsbruck.pdf, 2004.