Summer 2011

Improving the accuracy of the Generalized FDTD-Q scheme for solving the linear time-dependent Schrödinger equation

James John Elliot III

IMPROVING THE ACCURACY OF THE GENERALIZED FDTD-Q SCHEME FOR SOLVING THE LINEAR TIME-DEPENDENT SCHRÖDINGER EQUATION

by

James John Elliott, III, B.S., M.S.

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

COLLEGE OF ENGINEERING AND SCIENCE
LOUISIANA TECH UNIVERSITY

August 2011

LOUISIANA TECH UNIVERSITY

THE GRADUATE SCHOOL

07/15/2011

Date

We hereby recommend that the dissertation prepared under our supervision by James John Elliott, III

entitled

Improving the Accuracy of the Generalized FDTD-Q Scheme for Solving the Linear Time-Dependent Schrödinger Equation

be accepted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Computational Analysis and Modeling

Supervisor of Dissertation Research

Head of Department

Computational Analysis and Modeling

Department

Recommendation concurred in:

Advisory Committee

Approved:

Director of Graduate Studies

Approved:

Dean of the Graduate School

Dean of the College

GS Form 13a
(6/07)

ABSTRACT

This dissertation improves the accuracy of the Generalized Finite Difference Time Domain (FDTD) scheme by determining a differential operator that is capable of achieving reasonable accuracy when used to obtain even-order derivatives up to order fourteen. The Generalized FDTD scheme is an explicit scheme used to solve the time-dependent Schrodinger equation, and being an explicit scheme, it must utilize a carefully devised ratio of the temporal step to the spatial step to maintain numerical stability. This ratio is called the mesh ratio, and the Generalized FDTD scheme allows this ratio to be significantly relaxed. As the mesh ratio increases, the generalized scheme requires the evaluation of increasingly high-order spatial derivatives.

In Chapter 3, two classes of differential operators are considered, the first being the repeated application of a central difference approximation of the Laplace operator using various orders of accuracy, and the second class being the differentiated Lagrange interpolating polynomials. This approach, intentionally avoids attempting to approximate such derivatives using increasingly high-order finite differences, as the number of uncomputable points becomes very large as the order of the derivative increases.

Based on the conclusions from Chapter 3, a sixth-order accurate central difference operator is chosen to approximate the Laplace operator, and in Chapter 4 the order of accuracy is determined. The numerical stability is analyzed using the Von Neumann analysis and a stability condition is shown.

The validity of the analysis performed in Chapter 4 is verified by solving a

Schrödinger equation with exact solution, and observing the numerical error and stability. The order of accuracy of the scheme is also verified through experimentation, it is shown both theoretically and empirically that the chosen differential operator is both stable and accurate when used to solve the time-dependent Schrödinger equation using the Generalized FDTD method.

APPROVAL FOR SCHOLARLY DISSEMINATION

The author grants to the Prescott Memorial Library of Louisiana Tech University the right to reproduce, by appropriate methods, upon request, any or all portions of this Dissertation. It is understood that "proper request" consists of the agreement, on the part of the requesting party, that said reproduction is for his personal use and that subsequent reproduction will not occur without written approval of the author of this Dissertation. Further, any portions of the Dissertation used in books, papers, and other works must be appropriately referenced to this Dissertation.

Finally, the author of this Dissertation reserves the right to publish freely, in the literature, at any time, any or all portions of this Dissertation.

Author 

Date 08/05/2011

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I have been blessed by so many people in my life Through happiness and hardship my family has always been a constant place of support and encouragement I would like to thank my parents, James and Marie Elliott for instilling in me the work ethic and creativity that has made this possible

To my research advisers (and there are many) I must extend my heartfelt thanks Weizhong Dai and Galen Turner both provided more patience and support than I feel I would be capable of As I move forward in my career I feel I have met two of the best role models an aspiring professor could meet, and I can only hope that one day I can truly become their peer I would also like to thank Ramu Ramachandran for providing pivotal advice when I needed it most, and Chokchai Leangsuksun for introducing me to the field I have come to adore

A special thanks must also be given to Christian Duncan and Kelly Crittenden, who both provided detailed, meaningful feedback I must also thank David Mills, Daniela Mainardi, and Linda Ramsey for providing an excellent GK-12 Teaching Fellow program, in which not only did I teach but also learned

A final thanks goes to Jeff and Veronica Franks who have had such a profound impact on my life, and to Lisa James for her constant support throughout my graduate studies

CHAPTER 1

INTRODUCTION

In this chapter, a brief introduction to the time-dependent Schrödinger equation is provided, as well as an overview of the organization of this document, an introduction to the Taylor series and the Finite Difference method, and a summary of the motivation driving this work. Should the reader have a firm understanding of the topics listed, then they are encouraged to jump straight to Section 1.4 (page 10).

1.1 Time-Dependent Schrödinger Equation

The Schrödinger equation is a fundamental equation in quantum mechanics that describes how the wavefunction of a physical system evolves over time, predicting the behavior of a dynamic system. In the field of quantum mechanics, the wavefunction represents the quantum state and is the most complete description that can be given to a physical system. The one-dimensional time-dependent Schrödinger equation is

$$\frac{\partial \psi(x,t)}{\partial t} = \imath \frac{\hbar}{2m} \frac{\partial^2 \psi(x,t)}{\partial x^2} - i \frac{V(x,t)}{\hbar} \psi(x,t), \quad \text{for} \quad a \leq x \leq b \quad \text{and} \quad t > 0, \quad (1.1)$$

$$\psi(a,t) = \psi(b,t) = 0, \quad t > 0,$$

$$\psi(x,0) = \phi(x), \quad \text{for} \quad a \leq x \leq b,$$

where m is the mass of the particle [kg], $\hbar = 1.054 \times 10^{-34}$ [J-sec] is the reduced Planck's constant, $V(x,t)$ is a given real-valued potential function [J], $\psi(x,t)$ is a complex function, $\phi(x)$ is a complex initial condition, and $\imath$ is the imaginary unit

$\imath = \sqrt{-1}$ Since $\psi(x,t)$ is complex, then a specific value of ψ takes the general form $\psi(x,t) = \alpha + \imath\beta$, and the conjugate of $\psi(x\ t)$ is $\bar{\psi}(x,t) = \alpha - \imath\beta$, the product of the complex conjugates of $\psi(x,t)$ is then, $\psi(x,t)\ \ \bar{\psi}(x,t)$ indicating the probability of a particle being at the spatial location x at time t Obtaining a solution to the wavefunction ψ is a typical goal when solving the Schrodinger equation, and to do so requires solving a partial differential equation, which in this work is achieved using the Finite Difference Time Domain method When in the context of Quantum Mechanics, the abbreviation FDTD-Q is often used, and in this dissertation, FDTD and FDTD-Q will be used synonymously The focus of this work is centered on improving the accuracy of the approximation given by the FDTD method

1.2 Outline of the Dissertation

In this dissertation the Generalized Finite Difference Time Domain scheme as proposed by Dai and Moxley [11] is used, and after thorough analysis, improvements that lead to significantly more accurate solutions of the wavefunction ψ are proposed To begin, the FDTD-Q method and generalized FDTD-Q method are introduced in Chapter 2 In Chapter 3 two compelling methods for obtaining high-order derivatives are investigated, and after an introduction to the theory behind the methods, numerical investigations are performed and the results analyzed Following the conclusions from Chapter 3, improvements are proposed in Chapter 4, and the stability and accuracy of both the original method and modified method are analyzed

Combining the method selected in Chapter 3 for obtaining high-order derivatives and the considerations for stability and grid spacing presented in Chapter 2 numerical experiments are performed in Chapter 5 Here a model problem is solved allowing comparison with the exact analytical solution A practical simulation is also performed, where a particle moves through free-space and strikes an energy potential

1.3 Finite Differences and Taylor Series

Before moving forward there are a few crucial concepts that are used seamlessly throughout this dissertation, one such concept is the Taylor series and another is the Finite Difference method. These mathematical tools are not disjoint ideas, rather the Taylor series forms the foundation upon which the Finite Difference method is built, and it is for this reason they are both introduced here.

In the field of numerical analysis one of the most important tools and the foundation for the majority of the work presented here is the Taylor series, or the Taylor polynomials and associated truncation error. From Atkinson [3] we have Taylor's Theorem:

Theorem 1.1 (Taylor's Theorem). *Let $f(x)$ have $n+1$ continuous derivatives on $[a,b]$ for some $n \geq 0$, and let $x, x_0 \in [a.b]$. Then*

$$f(x) = T_n(x) + R_{n+1}(x)$$

$$T_n(x) = f(x_0) + \frac{(x-x_0)}{1!}f'(x_0) + \frac{(x-x_0)^2}{2!}f''(x_0) + \cdots + \frac{(x-x_0)^n}{n!}f^{(n)}(x_0)$$

$$R_{n+1}(x) = \frac{(x-x_0)^{n+1}}{(n+1)!}f^{(n+1)}(\xi)$$

for some ξ between x_0 and x.

When this expansion is convergent, that is $\lim_{n\to\infty} R_n = 0$, then the expansion is called the Taylor series of $f(x)$ expanded about x_0. Since n may not get arbitrarily large in practice, one typically truncates the Taylor series at some fixed n creating an *approximation* of the function $f(x)$, and when doing so the polynomial T_n is called the nth Taylor polynomial with the remainder term R_n called the *truncation error*. It is clear from Theorem 1.1 that if $|(x-x_0)|$ is sufficiently small i e., $|(x-x_0)| < 1$. then the truncation error will tend towards zero as n increases. Should $|(r-r_0)| \ll 1$ then the truncation error vanishes faster, and so based on n and $(x-x_0)$ one may

characterize the convergence rate of the truncated Taylor series $\frac{(x-x_0)}{(n+1)!}f^{(n+1)}(\xi)$ This characterization is called the *order of accuracy*

With the convergence rate characterized exclusively by n and $(x - x_0)$, a common notation is used that conveys this information clearly and concisely First, recognize that $(x - x_0) = \Delta x$, and now using Δx and n the order of accuracy may be expressed as $O(\Delta x^{n+1})$, which implies $O(\Delta x^{n+1}) = \frac{\Delta x^{n+1}}{(n+1)!}f^{(n+1)}(\xi)$

A caveat of using the Taylor series directly as presented is that one must have evaluations for the function f, as well as all derivatives up to the desired n When seeking solutions to partial differential equations one typically has function values $f(x)$, and wants to know the differentiated values One method for obtaining these differentiated values is called the Finite Difference method, and it works as follows

Suppose the solution to $f'(x)$ is desired, and one has solutions to $f(x)$ along a particular structured grid $\{(x_0\ f(x_0)),\quad ,(x_n, f(x_n))\}$ and x_i is defined as $x_i = i\Delta x$, for all $i \in \{0\quad ,n\}$ Then one may construct a Taylor series expansion for the function $f(x + \Delta x)$ about x as follows

$$T_n(x + \Delta x) = f(x + \Delta x) = f(x) + (x + \Delta x - x)f'(x) + \frac{(x + \Delta x - x)^2}{2!}f''(x)$$
$$+ \quad + \frac{(x + \Delta x - x)^n}{n!}f^{(n)}(x)$$
$$f(x + \Delta x) = f(x) + \Delta x f'(x) + \frac{\Delta x^2}{2}f''(x) + \quad + \frac{\Delta x^n}{n!}f^{(n)}(x) \qquad (1\ 2)$$

From Equation (1 2), the notation $T_n(x+\Delta x)$ is typically avoided, and when working with finite differences the name of the function being approximated is used It is understood that $f(x + \Delta x)$ is a Taylor series approximation Another shorthand notation common to the field of finite differences, arises from the fact that when working with a finite domain, where you have sequentially numbered x grid locations e g $x_0\ x_1\quad x_M$, then $f(x_k)$ may be shortened to $f(k) = f(x_k) = f_k$ In this writing the notation $f(k)$ is preferred

Returning to Equation (1 2) and solving for the desired derivative, which in this example is $f'(x)$,

$$-\Delta x f'(x) = f(x) - f(x+\Delta x) + \frac{\Delta x^2}{2} f''(x) + \quad + \frac{\Delta x^n}{n!} f^{(n)}(x)$$

$$f'(x) = \frac{f(x+\Delta x) - f(x)}{\Delta x} - \frac{\Delta x}{2} f''(x) - \quad - \frac{\Delta x^{n-1}}{n!} f^{(n)}(x) \tag{1 3}$$

From Equation (1 3) it is clear that if one only has function evaluations at $f(x)$, then f'' $f^{(3)}$, $\quad$, $f^{(n)}$ are all unknown terms, and so the Taylor series must be truncated

$$-\Delta x f'(x) = f(x) - f(x+\Delta x) + \frac{\Delta x^2}{2} f''(x) + \quad + \frac{\Delta x^n}{n!} f^{(n)}(x)$$

$$f'(x) = \frac{f(x+\Delta x) - f(x)}{\Delta x} + \frac{\Delta x}{2} f''(\xi(x))$$

$$f'(x) = \frac{f(x+\Delta x) - f(x)}{\Delta x} + O(\Delta x) \tag{1 4}$$

Equation (1 4) is called a *Forward Difference* and similarly there is a *Backward Difference* that may be constructed using a Taylor expansion of the function $f(x-\Delta x)$ about x

$$f(x-\Delta x) = f(x) - \Delta x f'(x) + \frac{\Delta x^2}{2} f''(x) + \quad + \frac{-\Delta x^n}{n!} f^{(n)}(x)$$

$$f'(x) = \frac{f(x) - f(x-\Delta x)}{\Delta x} + O(\Delta x) \tag{1 5}$$

Both approximations are first-order accurate $O(\Delta x)$, and to increase accuracy one must find a way to remove the higher-order derivatives leaving a higher-order remainder term One way to achieve this is intuitively which works well for differences with a small number of points but to derive more advanced schemes a more robust method is required One such method is the *Method of Undetermined Coefficients* First consider improving the above approximations of the first derivative by experimentation Rather than work with a single Taylor expansion, take two expansions one for $f(x+\Delta x)$ and another for $f(x-\Delta x)$ both centered

about x

$$f(x+\Delta x) = f(x) + f'(x)\Delta x + \frac{f''(x)\Delta x^2}{2!} + \frac{f^{(3)}(x)\Delta x^3}{3!} + \frac{f^{(4)}(x)\Delta x^4}{4!}$$
$$= + \frac{f^{(5)}(x)\Delta x^5}{5!} + \cdots + \frac{f^{(n)}(\xi)\Delta x^n}{n!}, \tag{1.6}$$
$$f(x-\Delta x) = f(x) - f'(x)\Delta x + \frac{f''(x)\Delta x^2}{2!} - \frac{f^{(3)}(x)\Delta x^3}{3!} + \frac{f^{(4)}(x)\Delta x^4}{4!}$$
$$= - \frac{f^{(5)}(x)\Delta x^5}{5!} + \cdots + \frac{f^{(n)}(\xi)\Delta x^n}{n!}. \tag{1.7}$$

Since the goal is to improve the truncation error which was previously $\frac{f''(x)\Delta x}{2!}$, one must remove the second derivative terms from the equations above. This may be achieved by subtracting Equation (1.7) from Equation (1.6)

$$f(x+\Delta x) - f(x-\Delta x) = 2f'(x)\Delta x$$
$$+ \cdots + \frac{f^{(2n+1)}(\xi)\Delta x^{2n+1}}{(2n+1)!} + \frac{f^{(2n+1)}(\eta)\Delta x^{2n+1}}{(2n+1)!} \tag{1.8}$$

Solving for the first derivative term,

$$2f'(x)\Delta x = f(x+\Delta x) - f(x-\Delta x) - \cdots - \frac{f^{(2n+1)}(\xi) + f^{(2n+1)}(\eta)}{(2n+1)!}\Delta x^{2n+1}$$
$$f'(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} - \cdots - \frac{f^{(2n+1)}(\xi) + f^{(2n+1)}(\eta)}{2(2n+1)!}\Delta x^{2n} \tag{1.9}$$
$$f'(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} - \frac{f^{(3)}(\xi) + f^{(3)}(\eta)}{2}\frac{\Delta x^2}{3!} \tag{1.10}$$

Recognizing that the third derivatives are unknowns leads to the *Central Difference* approximation of the first derivative with error term. In this case, the truncation error may be simplified based on the assumptions required by Taylor's Theorem, which state that $f(x)$ must have $n+1$ continuous derivatives on the interval $[a, b]$, and that both x and x_0 are in $[a, b]$. With these requirements, one may utilize the

Intermediate Value Theorem, taken from Atkinson [3], which states

Theorem 1.2 (Intermediate Value Theorem). *Let $f(x)$ be continuous on the finite interval $a \leq x \leq b$, and define*

$$m = \inf_{a \leq x \leq b} f(x), \qquad M = \sup_{a < x < b} f(x)$$

Then for any number u in the interval $[m, M]$, there is at least one point ξ in $[a, b]$ for which

$$f(\xi) = u$$

Or in simpler terms, the Intermediate Value Theorem is rigorously expressing the idea that for continuous functions, one must be able to draw a line (or curve) between $f(a)$ and $f(b)$ with out picking up the pencil When cast in the context of the above central difference remainder term, there must exist some μ such that

$$\frac{f^{(3)}(\xi) + f^{(3)}(\eta)}{2} = f^{(3)}(\mu),$$

and the second-order accurate central difference approximation of the first derivative is

$$f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} + \frac{f^{(3)}(\mu)}{3!}\Delta x^2$$

$$f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} + O(\Delta x^2) \tag{1.11}$$

In the previous examples, the coefficients of the Finite Difference scheme were determined directly by recognizing which terms should be added or subtracted from each other to make the unwanted terms vanish, but when attempting to construct a scheme that uses many points (function values) the previous method becomes increasingly difficult What follows is essentially the same method, but presented in a more robust manner This method is called the *Method of Undetermined Coefficients*

The general form of any central difference scheme is

$$Af(x-\Delta x)+Bf(x)+Cf(x+\Delta x),$$

$$Af(x-2\Delta x)+Bf(x-\Delta x)+Cf(x)+Df(x+\Delta x)+Ef(x+2\Delta x),$$

$$\vdots$$

$$C_0 f(x)+\sum_{k=1} C_{-k}f(x-k\Delta x)+C_{+k}f(x+k\Delta x) \qquad (1.12)$$

The goal is now to determine the coefficients $A, B.$ and C for a three point scheme, $A. B, C. D,$ and E for a five point scheme. If one looks at the Taylor expansions associated with the scheme, a linear system of equations may be formed. For example, consider increasing the accuracy of the previous second-order accurate central difference approximation of the first derivative Equation (1.11). First assume the form of the final central difference scheme will be $Af(x+2\Delta x)+Bf(x+\Delta x)+Cf(x)+Df(x-\Delta x)+Ef(x-2\Delta x)$, and construct Taylor expansions as in the previous examples

$$f(x+\Delta x)=f(x) \quad +f'(x)\Delta x+ \quad \frac{f''(x)\Delta x^2}{2!}+\cdots+\frac{f^{(n)}(\xi)\Delta x^n}{n!},$$

$$f(x+2\Delta x)=f(x) \quad +2f'(x)\Delta x+ \quad 2^2\frac{f''(x)\Delta x^2}{2!}+\cdots+2^n\frac{f^{(n)}(\gamma)\Delta x^n}{n!}.$$

$$f(x-\Delta x)=f(x) \quad -f'(x)\Delta x+ \quad \frac{f''(x)\Delta x^2}{2!}+\cdots+\frac{f^{(n)}(\eta)(-\Delta x)^n}{n!},$$

$$f(x-2\Delta x)=f(x) \quad -2f'(x)\Delta x+ \quad 2^2\frac{f''(x)\Delta x^2}{2!}+ \quad \cdot+2^n\frac{f^{(n)}(\zeta)(-\Delta x)^n}{n!}.$$

Now multiply each Taylor expansion by its associated unknown coefficient

$$Bf(x+\Delta x)=Bf(x) \quad +Bf'(x)\Delta x+B\frac{f''(x)\Delta x^2}{2!}+\cdots+B\frac{f^{(n)}(\xi)\Delta x^n}{n!},$$

$$Af(x+2\Delta x)=Af(x) \quad +2Af'(x)\Delta x+2^2A\frac{f''(x)\Delta x^2}{2!}+\cdots+2^nA\frac{f^{(n)}(\gamma)\Delta x^n}{n!},$$

$$Df(x-\Delta x)=Df(x) \quad -Df'(x)\Delta x+D\frac{f''(x)\Delta x^2}{2!}+\cdots+D\frac{f^{(n)}(\eta)(-\Delta x)^n}{n!}$$

$$Ef(x-2\Delta x) = Ef(x) \quad -2Ef'(x)\Delta x + 2^2 E\frac{f''(x)\Delta x^2}{2!} + \cdots + 2^n E\frac{f^{(n)}(\zeta)(-\Delta x)^n}{n!}.$$

and form a linear system of equations such that the coefficient for f' must be 1, and the coefficient for all other terms must be zero. The goal is to isolate the derivative in the Taylor expansion we wish to approximate, while providing sufficient information such that the linear system is solvable, in this case five equations and five unknowns The form of the desired solution is

$$Af(r+2\Delta r) + bf(r+\Delta r) + Cf(x) + Df(x-\Delta r) + Ef(r-2\Delta r) = T(r), \quad (1.13)$$

and $T(x)$ represents the truncated Taylor series resulting from the left hand side of Equation (1 13) Thus,

$$T(x) = C_0 f(x) + C_1 f'(x)\Delta x + C_2\frac{f''(x)}{2!}\Delta x^2 + C_3\frac{f^{(3)}(x)}{3!}\Delta x^3 + C_4\frac{f^{(4)}(x)}{4!}\Delta r^4 + C_\phi\frac{f^{(5)}(\phi)}{5!}\Delta x^5 \quad (1.14)$$

Equation (1.13) and Equation (1.14) together imply that the following equations must be true

$$\text{Equation for } C_0 f(x) \qquad (A+B+C+D+E) = 0 = C_0, \quad (1\ 15a)$$

$$\text{Equation for } C_1 f^{(1)}(x), \qquad (A+2B-D-2E) = \frac{1}{\Delta x} = C_1, \quad (1.15b)$$

$$\text{Equation for } C_2 f^{(2)}(x), \qquad (A+2^2B+D+2^2E) = 0 = C_2. \quad (1.15c)$$

$$\text{Equation for } C_3 f^{(3)}(x), \qquad (A+2^3B-D-2^3E) = 0 = C_3. \quad (1.15d)$$

$$\text{Equation for } C_4 f^{(4)}(r), \qquad (A+2^4B+D+2^4E) = 0 = C_4 \quad (1.15e)$$

It is considerably easier to represent this technique in matrix form

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 2 \\ 4 & 1 & 0 & 1 & 4 \\ -8 & -1 & 0 & 1 & 8 \\ 16 & 1 & 0 & 1 & 16 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{\Delta x} \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

which when solved, yields

$$A = -\frac{1}{12\Delta x}, \qquad B = \frac{2}{3\Delta x}, \qquad C = 0, \qquad D = -\frac{2}{3\Delta x}, \qquad E = \frac{1}{12\Delta x},$$

or in the more common form

$$f'(x) = \frac{-f(x+2\Delta x) + 8f(x+\Delta x) - 8f(x-\Delta x) + f(x-2\Delta x)}{12\Delta x} + O(\Delta x^4) \tag{1.16}$$

Following the same procedure one may create Finite Difference schemes that approximate any derivative up to any order of accuracy Of particular interest to this work are central difference approximations of the second derivative, which are then used to approximate the Laplace operator $\nabla^2 = \frac{\partial^2}{\partial x^2}$ in one dimension

1.4 Motivation

Without delving too far into the Generalized FDTD-Q method, sufficient background is presented here so that one may still appreciate its improvements over the traditional FDTD-Q method As the name implies the FDTD-Q method is the Finite Difference Time Domain method applied to quantum mechanics Finite Difference Time Domain schemes follow the same fundamental ideas presented in the previous section Specifically they employ various differencing operators on both space and time which means FDTD schemes may have various orders of accuracy In general the order of accuracy of an FDTD scheme is a function of both the spatial step and temporal step such as $O(\Delta x^2 + \Delta t)$ or in two dimensions $O(\Delta x^2 +$

$\Delta y^2 + \Delta t^2$). The original FDTD-Q method for solving the Schrödinger equation has a proven accuracy of $O(\Delta x^2 + \Delta x^2 \Delta t^2 + \Delta t^2)$, and the generalized FDTD-Q method uses additional Taylor series expansions in time that allow one to achieve very high-orders of accuracy in time, such as $O(\Delta x^2 + \Delta x^2 \Delta t^2 + \cdots + \Delta t^{(2N+3)})$, where N is a parameter related to the number of derivative terms in the Taylor expansion that are evaluated ($N = 0$ yields the original FDTD-Q method).

The Generalized FDTD-Q scheme as published [11] provides second-order spatial accuracy, while providing arbitrarily high time accuracy. The reason the Generalized FDTD-Q scheme has only been able to achieve second-order spatial accuracy, is the requirement that the scheme with parameter N requires all even number spatial derivatives from $2.4, \ldots, 4N + 2$. Obtaining these high-order derivatives has proven challenging for two specific reasons that are directly addressed in this dissertation. First, the accuracy of the derivatives degrades rapidly as the order of the derivative increases. Second, attempting to use methods with higher accuracy than the second-order accurate central difference leads to stability issues that must be addressed.

CHAPTER 2

REVIEW OF THE FDTD-Q METHODS

The one-dimensional (1-D) time-dependent linear Schrödinger equation, was introduced in Chapter 1, Equation (1.1). In this Chapter, a survey of previous work will be introduced, as well as a general overview of what makes the Generalized FDTD-Q method novel.

Before proceeding further, it should be understood that there are two main types of Finite Difference Time Domain schemes. The first type are called Explicit schemes, as it is possible to compute the solution at time $n+1$ directly using only information from previous time steps. That is to say, with explicit schemes it is possible to formulate the problem such that there is a single unknown on the left-hand side of the equation and all known values on the right-hand side. The other type of scheme is an Implicit scheme, which means that the solution to the current time step is obtained by solving a system of equations based on previous and future time steps.

There are strengths and weaknesses to each type of scheme. The implicit schemes are unconditionally stable, meaning one may choose the time step independent of the choice of the spatial step, but the cost of the implicit scheme is that a system of equations must be solved at each time step and the equations are typically more complex than those in an explicit scheme. Explicit schemes are typically easier to compute, but are not unconditionally stable. This means that there is a restriction imposed on the mesh ratio $\frac{\Delta t}{\Delta x^2} < c$, where devising a method that allows one to relax

this restriction is optimal, as you obtain the ability to not only compute solutions directly but also move through time faster.

2.1 FDTD-Q Methods

Many numerical schemes have been developed for solving linear Schrödinger equations [1,2,4–7,9,10,12–21,23,28,30–33]. Of the works [1,2,4–7,9,10,12–21,23,28, 30–33], it should be noted which type of method each used to solve the Schrödinger equation. Of those works, the ones that utilized a method that required the solution of a matrix are [1, 4–7, 12–18, 20, 21, 23, 30–33], which is clearly the majority of the previous work. Sullivan [29], in his book on electromagnetic simulations, extended the ideas used to solve Maxwell's equation using the FDTD method, to solve the linear Schrödinger equation using an explicit scheme. From his book, the formulation of the explicit FDTD scheme is as follows:

To avoid the use of complex numbers, the wavefunction ψ is split into its real and imaginary components,

$$\psi(x.t) = \psi_{\text{real}}(x,t) + \imath\psi_{\text{imag}}(x.t). \tag{2.1}$$

Inserting Equation (2.1) into Equation (1.1) and then separating the real and imaginary parts result in the following coupled set of equations:

$$\frac{\partial\psi_{\text{real}}(x.t)}{\partial t} = -\frac{\hbar}{2m}\frac{\partial^2\psi_{\text{imag}}(x,t)}{\partial x^2} + \frac{V(x,t)}{\hbar}\psi_{\text{imag}}(x,t) \tag{2.2a}$$

and

$$\frac{\partial\psi_{\text{imag}}(x,t)}{\partial t} = \frac{\hbar}{2m}\frac{\partial^2\psi_{\text{real}}(x,t)}{\partial x^2} - \frac{V(x,t)}{\hbar}\psi_{\text{real}}(x,t). \tag{2.2b}$$

Thus, the second-order accurate finite difference approximations in space and time result in the FDTD scheme as follows:

$$\frac{\psi^n_{\text{real}}(k) - \psi^{n-1}_{\text{real}}(k)}{\Delta t} = -\frac{\hbar}{2m\Delta x^2}\delta_x^2\psi^{n-\frac{1}{2}}_{\text{imag}}(k) + \frac{1}{\hbar}V(k)\psi^{n-\frac{1}{2}}_{\text{imag}}(k) \tag{2.3a}$$

and

$$\frac{\psi_{\text{imag}}^{n+\frac{1}{2}}(k) - \psi_{\text{imag}}^{n-\frac{1}{2}}(k)}{\Delta t} = \frac{\hbar}{2m\Delta x^2}\delta_x^2\psi_{\text{real}}^n(k) - \frac{1}{\hbar}V(k)\psi_{\text{real}}^n(k) \tag{2 3b}$$

In the transition from Equation (2 2) to Equation (2 3), the notation has clearly changed This change is a direct result of discretizing the analytical form in Equation (2 2) into a discrete form in Equation (2 3) The spatial domain x and temporal domain t have been discretized into a finite set of equally spaced grid locations,

$$x = a + k\Delta x, \quad \text{for} \quad a \leq x \leq b, k = 0 \quad , \frac{b-a}{\Delta x} \tag{2 4}$$

and

$$t = n\Delta t, \quad \text{for} \quad t > 0, n = 1, \quad , N_{\text{steps}} \tag{2 5}$$

The functions ψ_{real} and ψ_{imag} are then solved at the grid locations from Equation (2 4) at a specific time step n The notation should then be interpreted as

$$\psi_{\text{real}}^n(k) = \psi_{\text{real}}(x_k, t_n) = \psi_{\text{real}}(a + k\Delta x, n\Delta t)$$

and

$$\psi_{\text{imag}}^{n+\frac{1}{2}}(k) = \psi_{\text{imag}}(x_k, t_{n+\frac{1}{2}}) = \psi_{\text{imag}}(a + k\Delta x, (n + \frac{1}{2})\Delta t)$$

Equation (2 3) then becomes the starting point for attempting to improve the explicit FDTD scheme, as the form shown by Sullivan is second-order accurate in space and time $O(\Delta t^2 + \Delta x^2\Delta t^2)$, but no stability condition was known From this point two independent researchers [10, 28] reached nearly identical bounds for the stability of the explicit FDTD scheme Dai *et al* [10] used the discrete energy method to show that scheme is stable if

$$\frac{\hbar}{m}\left[\frac{\Delta t}{\Delta x^2} + \frac{\Delta t}{2\hbar}\max|V|\right] \leq c < 1 \tag{2 6}$$

where c is a constant Soriano *et al* [28] used the eigenvalue method to analyze the

stability of the FDTD scheme and obtained a very similar condition of

$$\frac{\hbar}{m} \cdot \frac{\Delta t}{\Delta x^2} + \frac{\Delta t}{2\hbar} \max |V| \leq 1. \tag{2.7}$$

The above stability conditions are imperative to this work, as we will utilize the bound shown by Dai [10] to construct a specific stability condition for the scheme proposed. Dai [10] noted that even if the condition shown by Soriano is chosen, the numerical solution may still diverge, and that the stability condition in Equation (2.6) indicates that the condition for stability must be less than one but not close to one.

This leads to the motivation for creating the Generalized FDTD-Q method as proposed by Dai and Moxley [11], which is to relax the restriction on the mesh ratio, $\frac{\Delta t}{\Delta x^2}$.

2.2 Generalized FDTD-Q Method

The Generalized FDTD-Q scheme proposed by Dai and Moxley [11] is an integral part of this dissertation. In this section a brief overview of how the method was derived is shown, because it will be utilized frequently in Chapter 4. But more important than the derivation is to realize that the method provides arbitrary accuracy in time and theoretical unconditional stability. The cost for obtaining higher accuracy in time, is that one must be able to evaluate high-order spatial derivatives.

To develop the Generalized FDTD-Q scheme, one must assume that $\psi_{\text{real}}(x,t)$ and $\psi_{\text{imag}}(x,t)$ are sufficiently smooth functions which vanish for sufficiently large $|x|$ and the potential V is dependent only on x [11]. The scheme is then derived as shown in [11], which is summarized here. Eqs. (2.2a) and (2.2b) are rewritten as

$$\frac{\partial \psi_{\text{real}}(x,t)}{\partial t} = (-\frac{\hbar}{2m} A + \frac{V}{\hbar}) \psi_{\text{imag}}(x,t), \tag{2.8a}$$

$$\frac{\partial \psi_{\text{imag}}(x,t)}{\partial t} = (\frac{\hbar}{2m} A - \frac{V}{\hbar}) \psi_{\text{real}}(x,t). \tag{2.8b}$$

where $A = \frac{\partial^2}{\partial x^2}$. The real component of the wavefunction is then expanded using the Taylor series at $\psi_{\text{real}}(x.t_n)$ and $\psi_{\text{real}}(x, t_{n-1})$ about $t = t_{n-\frac{1}{2}} = (n - \frac{1}{2})\Delta t$, and the imaginary component is expanded at $\psi_{\text{imag}}(t_{n+\frac{1}{2}})$ and $\psi_{\text{imag}}(t_{n-\frac{1}{2}})$ about $t = t_n$. Using the relations in Equation (2.8a) and Equation (2.8b) the resulting derivatives may be evaluated leading to the Generalized FDTD-Q scheme

$$\psi^n_{\text{real}}(k) = \psi^{n-1}_{\text{real}}(k) + 2\sum_{p=0}^{N}(\frac{\Delta t}{2})^{2p+1}\frac{(-1)^{p+1}}{(2p+1)!}(\frac{\hbar}{2m}A - \frac{V}{\hbar})^{2p+1}\psi^{n-\frac{1}{2}}_{\text{imag}}(k), \tag{2.9a}$$

$$\psi^{n+\frac{1}{2}}_{\text{imag}}(k) = \psi^{n-\frac{1}{2}}_{\text{imag}}(k) + 2\sum_{p=0}^{N}(\frac{\Delta t}{2})^{2p+1}\frac{(-1)^{p}}{(2p+1)!}(\frac{\hbar}{2m}A - \frac{V}{\hbar})^{2p+1}\psi^{n}_{\text{real}}(k), \tag{2.9b}$$

which depends greatly on the ability to accurately approximate the Laplace operator A. And it is the approximation of this operator which is the motivation of this work. Following in the steps of [6, 21, 26, 34] we explore two compelling approximations of the Laplace operator. one utilizing various differentiated Lagrange polynomials and another using various central difference approximations.

CHAPTER 3

NUMERICAL DIFFERENTIATION

In this chapter an introduction to numerical differentiation is presented with a specific focus on two methods, the first being the central difference method and the second the differentiated Lagrange interpolating polynomials Each method is compared and a conclusion is drawn based on error propagation and computational complexity

3.1 Central Difference Approximations

Having covered a number of fundamental preliminaries in Chapter 1 Section 1 3, this section will begin with the generation of highly accurate approximations of the Laplace operator For the scope of this section we shall assume one has some function $f(x)$ that has seven continuous derivatives over the interval $[a, b]$, and one has solutions to $f(x)$ along a particular structured grid $\{(x_0, f(x_0)), \quad , (x_n, f(x_n))\}$ where x_i is defined as $x_i = i\Delta x$ for all $i \in \{0, \quad , n\}$

First consider a central difference scheme that takes the form

$$Af(x - \Delta x) + Bf(x) + Cf(x + \Delta x) = f''(x) + O(\Delta x^2) \tag{3 1}$$

Figure 3 1 illustrates the stencil this differencing scheme uses Note that the grid points x_0 and x_n are unsolvable points since x_{-1} and x_{n+1} do not exist This inherent limitation of the central differences can cause problems at the boundaries where the accuracy tends to degrade

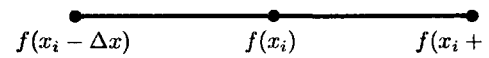


Figure 3.1: The three point central difference stencil which gives second-order accurate approximations of the second derivative.

Following the Method of Undetermined Coefficients presented in Section 1.3, one first obtains the Taylor expansions about x

$$Cf(x+\Delta x) = Cf(x) + Cf'(x)\Delta x + C\frac{f''(x)\Delta x^2}{2!} + \cdots + C\frac{f^{(n)}(\xi)\Delta x^n}{n!},$$
$$Af(x-\Delta x) = Af(x) - Af'(x)\Delta x + A\frac{f''(x)\Delta x^2}{2!} + \cdots + A\frac{f^{(n)}(\eta)(-\Delta x)^n}{n!},$$

and since we seek a second-order accurate approximation we require

$$T(x) = C_0 f(x) + C_1 f'(x)\Delta x + C_2\frac{f''(x)}{2!}\Delta x^2 + C_3\frac{f^{(3)}(x)}{3!}\Delta x^3 + C_\phi\frac{f^{(4)}(\phi)}{4!}\Delta x^4. \tag{3.2}$$

This implies that the following equations must be satisfied

$$\text{Equation for } C_0 f(x), \qquad (A+B+C) = 0 = C_0, \tag{3.3}$$
$$\text{Equation for } C_1 f^{(1)}(x), \qquad (C-A) = 0 = C_1, \tag{3.4}$$
$$\text{Equation for } C_2 f^{(2)}(x), \qquad (C+A) = \frac{2}{\Delta x^2} = C_2, \tag{3.5}$$
$$\text{Equation for } C_3 f^{(3)}(x), \qquad (C-A) = 0 = C_3. \tag{3.6}$$

Note that Equation (3.4) and Equation (3.6) are identical, or more precisely. Equation (3.6) is a linear combination of Equation (3.4) and zero, which means Equation (3.6) may be removed from the linear system. The resulting matrix is then

$$\begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{2}{\Delta x^2} \end{bmatrix},$$

which when solved, yields

$$A = \frac{1}{\Delta x^2}, \qquad B = \frac{-2}{\Delta x^2}, \qquad C = \frac{1}{\Delta x^2}$$

or in the more common form

$$f''(x) = \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2} + O(\Delta x^2). \tag{3.7}$$

Next, suppose one wishes to improve the accuracy, and so assume

$$Af(x+2\Delta x) + Bf(x+\Delta x) + Cf(x) + Df(x-\Delta x) + Ef(x-2\Delta x)$$
$$= f''(x) + O(\Delta x^4) = T(x), \tag{3.8}$$

and construct the required Taylor expansions about x.

$$Af(x+2\Delta x) = Af(x) \quad +2Af'(x)\Delta x + 2^2 A\frac{f''(x)\Delta x^2}{2!} + \cdots + 2^n A\frac{f^{(n)}(\gamma)\Delta x^n}{n!},$$

$$Bf(x+\Delta x) = Bf(x) \quad +Bf'(x)\Delta x + B\frac{f''(x)\Delta x^2}{2!} + \cdots + B\frac{f^{(n)}(\xi)\Delta x^n}{n!}$$

$$Df(x-\Delta x) = Df(x) \quad -Df'(x)\Delta x + D\frac{f''(x)\Delta x^2}{2!} + \cdots + D\frac{f^{(n)}(\eta)(-\Delta x)^n}{n!}.$$

$$Ef(x-2\Delta x) = Ef(x) \quad -2Ef'(x)\Delta x + 2^2 E\frac{f''(x)\Delta x^2}{2!} + \cdots + 2^n E\frac{f^{(n)}(\zeta)(-\Delta x)^n}{n!}$$

The Taylor polynomial must have the form

$$T(x) = C_0 f(x) + C_1 f'(x)\Delta x + C_2\frac{f''(x)}{2!}\Delta x^2 + C_3\frac{f^{(3)}(x)}{3!}\Delta x^3$$
$$+ C_4\frac{f^{(4)}(x)}{4!}\Delta x^4 + C_5\frac{f^{(5)}(x)}{5!}\Delta x^5 + C_\phi\frac{f^{(6)}(\phi)}{5!}\Delta x^6, \tag{3.9}$$

implying that the following system of equations must hold

$$\text{Equation for } C_0 f(x), \qquad (A+B+C+D+E) = 0 = C_0, \tag{3.10a}$$

$$\text{Equation for } C_1 f^{(1)}(x). \qquad (2A+B-D-2E) = 0 = C_1. \tag{3.10b}$$

$$\text{Equation for } C_2 f^{(2)}(x), \qquad (2^2 A+B+D+2^2 E) = \frac{2}{\Delta x^2} = C_2. \tag{3.10c}$$

$$\text{Equation for } C_3 f^{(3)}(x), \qquad (2^3 A + B - D - 2^3 E) = 0 = C_3, \tag{3.10d}$$

$$\text{Equation for } C_4 f^{(4)}(x), \qquad (2^4 A + B + D + 2^4 E) = 0 = C_4, \tag{3.10e}$$

$$\text{Equation for } C_5 f^{(5)}(x), \qquad (2^5 A + B - D - 2^5 E) = 0 = C_5. \tag{3.10f}$$

The system of linear of equations, Equation (3.10a) to Equation (3.10f) clearly lead to

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 0 & -1 & -2 \\ 4 & 1 & 0 & 1 & 4 \\ 8 & 1 & 0 & -1 & -8 \\ 16 & 1 & 0 & 1 & 16 \\ 32 & 1 & 0 & -1 & -32 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{2}{\Delta x^2} \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

which may be reduced recognizing that row six, denoted as R_6, of the above matrix is a linear combination of rows two and four, denoted as R_2 and R_4 respectively, $-4R_2+5R_4 = R_6$. Solving the simplified linear system yields the following coefficients

$$A = -\frac{1}{12\Delta x^2}, \quad B = \frac{4}{3\Delta x^2}, \quad C = -\frac{5}{2\Delta x^2}, \quad D = \frac{4}{3\Delta x^2}, \quad E = -\frac{1}{12\Delta x^2},$$

or

$$f''(x) = \frac{-f(x+2\Delta x) + 16f(x+\Delta x) - 30f(x) + 16f(x-\Delta x) - f(x-2\Delta x)}{12\Delta x^2} + O(\Delta x^4). \tag{3.11}$$

The stencil for this fourth-order accurate central difference approximation of the second derivative takes the form shown in Figure 3.2.

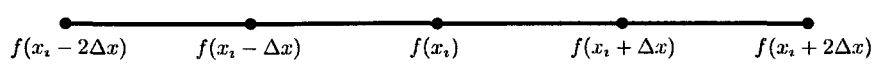


Figure 3.2: The five point central difference stencil which gives fourth-order accurate approximations of the second derivative.

Note that when $i \in \{0, 1, n-1, n\}$ this scheme is unsolvable, and the values at these points must either be guessed or approximated using some other technique. Following the same procedure one final sixth-order accurate central difference approximation of the second derivative is generated. Again assume the central difference will take the form

$$Af(x+3\Delta x) + Bf(x+2\Delta x) + Cf(x+\Delta x) + Df(x) + Ef(x-\Delta x)$$
$$+ Ff(x-2\Delta x) + Gf(x-3\Delta x) = f''(x) + O(\Delta x^6) = T(x). \tag{3.12}$$

After using the Method of Undetermined Coefficients, and for brevity, omitting the $\frac{1}{\Delta x^2}$ factor, one will arrive at

$$A = \frac{1}{90}, \quad B = -\frac{3}{20}, \quad C = \frac{3}{2}, \quad D = -\frac{49}{18}, \quad E = \frac{3}{2} \quad F = -\frac{3}{20} \quad G = \frac{1}{90},$$

which may be written in the standard form

$$f''(x) = \frac{1}{180\Delta x^2}\Big[2f(x+3\Delta x) - 27f(x+2\Delta x) + 270f(x+\Delta x) - 490f(x)$$
$$+ 270f(x-\Delta x) - 27f(x-2\Delta x) + 2f(x+3\Delta x)\Big] + O(\Delta x^6). \tag{3.13}$$

The graphical depiction of the stencil is shown in Figure 3.3. Again take note that while the accuracy of the approximations has increased, the restrictions on computing values near the boundary has grown, with $i \in \{0, 1, 2, n-2, n-1, n\}$ becoming uncomputable.

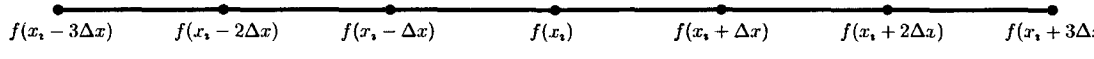


Figure 3.3: The seven point central difference stencil which gives sixth-order accurate approximations of the second derivative.

In the next section, a method that is *not* based on finite differences of Taylor series is derived. Here we will explore the use of Lagrange interpolating polynomials to approximate various derivatives. The motivation for doing this is simple, to obtain

accurate approximations of the derivatives at all points in the domain, whereas the central difference methods provide high accuracy with the cost that as the accuracy increases so do the number of uncomputable points on the grid.

3.2 Lagrange Interpolation

Rather than use Taylor series, this work will focus on using Lagrange polynomials, which are distinctly different from Taylor series in that the polynomial is constructed using information from all points in the domain rather than being centered about a specific point. A challenge of this work is that the error term associated with the Lagrange method increases both with the order of the derivative as well as with the number of grid points used in the interpolation. This is contrary to the Taylor methods which have the property that if Δx is chosen correctly, the truncation error tends toward zero as the n grows arbitrarily large.

To begin, consider the general form of the Lagrange interpolating polynomial and associated error term

$$f(x) = \sum_{j=0}^{n} w_j(x) f(x_j) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{k=0}^{n} (x - x_k). \tag{3.14}$$

Here $w_j(x)$ is a weighting function, and more specifically, $w_j(x)$ will form a Lagrange basis polynomial

$$w_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{(x - x_k)}{(x_j - x_k)}. \tag{3.15}$$

The function $f(x)$ is then approximated as a linear combination of Lagrange basis polynomials and associated function values $f(x_j)$. It is important to realize that the function values $f(x_j)$ are known values.

3.2.1 Properties of the Lagrange Basis Polynomials

Assume that one seeks to create a Lagrange interpolating polynomial using some given abscissas $x \in \{x_0, \ldots, x_n\}$ and associated function values $f(x_i)$ for all $i \in$

$\{0, \ldots, n\}$, and suppose one only seeks to evaluate this polynomial at the abscissas $x \in \{x_0, \ldots, x_n\}$. Under these assumptions there are two intriguing properties to note from the Lagrange basis polynomials that are directly relevant to this work. First, consider the case of $w_j(x_i)$, where $j \neq i$

$$w_j(x_i) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{(x_i - x_k)}{(x_j - x_k)} = \frac{(x_i - x_0)}{(x_j - x_0)} \cdots \frac{(x_i - x_i)}{(x_j - x_i)} \cdots \frac{(x_i - x_n)}{(x_j - x_n)} \quad \text{for} \quad j \neq i, \tag{3.16}$$

$$w_j(x_i) = 0 \quad \text{for} \quad j \neq i. \tag{3.17}$$

The numerator contains a term when $i = k$, which causes the entire expansion to become zero. Applying a similar analysis to the error term in Equation (3.14), when seeking the solution at x_i, where x_i is also one of the points used to construct the polynomial, the error term will become zero.

$$f(x_i) = P(x_i) + \frac{f^{(n+1)}(\xi(x_i))}{(n+1)!} \prod_{k=0}^{n} (x_i - x_k), \tag{3.18}$$

$$f(x_i) = P(x_i) + \frac{f^{(n+1)}(\xi(x_i))}{(n+1)!} (x_i - x_0) \cdots (x_i - x_i) \cdots (x_i - x_n), \tag{3.19}$$

$$f(x_i) = P(x_i). \tag{3.20}$$

Now consider the case where $i = j$,

$$w_j(x_i) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{(x_i - x_k)}{(x_i - x_k)} = 1 \quad \text{for} \quad j = i. \tag{3.21}$$

The resulting linear combination will take the form

$$f(x_i) = \sum_{j=0}^{n} w_j(x_i) f(x_j), \tag{3.22}$$

$$f(x_i) = w_0(x_i) f(x_0) + \cdots + w_i(x_i) f(x_i) + \cdots + w_n(x_i) f(x_n), \tag{3.23}$$

$$f(x_i) = 0 + 0 + \quad + 0 + f(x_i) + 0 + \quad + 0, \tag{3 24}$$

$$f(x_i) = f(x_i) \quad \text{for all} \quad x_i \in \{x_0, \quad x_n\}, \tag{3 25}$$

hence, the Lagrange interpolating polynomial exactly interpolates the function f

3.2.2 Differentiating Lagrange Interpolating Polynomials

Having introduced Lagrange polynomial interpolation, we will now focus on how to obtain derivative approximations To do this the Lagrange interpolating polynomial will be analytically differentiated Starting from Equation (3 14), and differentiating with respect to x yields

$$f'(x) = P'(x) + \frac{d}{dx}\left[\frac{f^{(n+1)}(\xi(x))}{(n+1)!}\prod_{k=0}^{n}(x - x_k)\right] \tag{3 26}$$

Focusing on the error term first, and applying the product rule for derivatives yields

$$\begin{aligned}\frac{d}{dx}\left[\frac{f^{(n+1)}(\xi(x))}{(n+1)!}\prod_{k=0}^{n}(x - x_k)\right] = \quad & \frac{d}{dx}\left[\frac{f^{(n+1)}(\xi(x))}{(n+1)!}\right]\prod_{k=0}^{n}(x - x_k) \\ & + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}\frac{d}{dx}\left[\prod_{k=0}^{n}(x - x_k)\right],\end{aligned} \tag{3 27}$$

and letting

$$Q(x) = \frac{d}{dx}\left[\frac{f^{(n+1)}(\xi(x))}{(n+1)!}\right]\prod_{k=0}^{n}(x - x_k), \text{ and}$$

$$R(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}\frac{d}{dx}\left[\prod_{k=0}^{n}(x - x_k)\right]$$

Returning to the original assumptions made at the beginning of Section 3 2 1, which are that r will be chosen from the same locations being interpolated that is $r \in \{x_0 \quad x_n\}$ and $i\ j\ k \in \{0 \quad n\}$ From this assumption, evaluating $Q(x)$ at any $x = x_i$ will force exactly one term to be $x_i = x_k$, yielding a zero in the product expansion forcing $Q(x_i)$ to be zero and simplifying the analysis of the error term for the first derivative Expanding $R(i)$ and repeatedly applying the product rule leads

to

$$\frac{f^{(n+1)}(\xi(x))}{(n+1)!}\frac{d}{dx}\left[\prod_{i=0}^{n}(x-x_i)\right] = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}\sum_{j=0}^{n}\prod_{\substack{k=0\\k\neq j}}^{n}(x-x_k). \tag{3.28}$$

Again, consider evaluating $R(x)$ at $x = x_i$. All but one product expansion in the summation will be zero because i will equal k. The remaining product expansion will exist, because $i = j$ and x_j was excluded in that specific product. The resulting Equation (3.29) is the form of the first derivative of a Lagrange interpolating polynomial and associated error

$$f'(x_i) = P'(x_i) + \frac{f^{(n+1)}(\xi(x_i))}{(n+1)!}\prod_{\substack{k=0\\k\neq i}}^{n}(x_i - x_k). \tag{3.29}$$

Note that unlike the original Lagrange interpolating polynomial, the differentiated Lagrange polynomial contains an error term that does not go to zero when evaluated at an interpolation point.

3.2.3 Differentiated Lagrange Weight Function

Having differentiated the Lagrange error term in detail, the derivative of the weight function $w_j(x)$ will be briefly discussed. Quan and Chang [24, 25] published a useful algorithm for computing the first derivative weights, and Shu [27] published a scheme for computing the weights for any derivative. The general form of the first derivative of the weight function is

$$w_j'(x_i) = \frac{1}{x_j - x_i}\prod_{\substack{k=0\\k\neq i,j}}^{n}\frac{(x_i - x_k)}{(x_j - x_k)}. \tag{3.30}$$

which may be obtained by repeated applications of the product rule. Recognizing the special case when $w_j(x_i)$ and $j = i$, allows one to arrive at the form proposed by Quan and Chang [25]

$$w'_j(x_i) = \frac{1}{x_j - x_i} \prod_{\substack{k=0 \\ k\neq i,j}}^{n} \frac{(x_i - x_k)}{(x_j - x_k)} \qquad \text{for} \quad i \neq j. \tag{3.31a}$$

$$w'_j(x_i) = \sum_{\substack{k=0 \\ k\neq i}}^{n} \frac{1}{x_i - x_k} \qquad \text{for} \quad i = j. \tag{3.31b}$$

Obtaining second derivative weights requires analytically differentiating the weight function $w_j(x)$ twice, which leads to the form published initially by Quan and Chang [24, 25] and again by Shu [27]

$$w''_j(x_i) = \frac{2}{x_j - x_i} \prod_{\substack{k=0 \\ k\neq i,j}}^{n} \frac{(x_i - x_k)}{(x_j - x_k)} \sum_{\substack{l=0 \\ l\neq i,j}}^{n} \frac{1}{x_i - x_l}, \qquad \text{for} \quad i \neq j, \tag{3.32a}$$

$$w''_j(x_i) = 2 \sum_{\substack{k=0 \\ k\neq i}}^{n-1} \frac{1}{x_i - x_k} \sum_{\substack{l=k \\ l\neq i}}^{n} \frac{1}{x_i - x_l}, \qquad \text{for} \quad i = j. \tag{3.32b}$$

Building off Quan and Chang's work, Shu [27] developed a recursive formula for computing higher-order differentiated weight functions, requiring only that the first derivative weights be computed using Equation (3.31). This method is superior to analytically differentiating the weights multiple times. as each differentiation requires an additional summation. Shu's method is as follows, where $w_j^{(1)}(x_i) = w'_j(x_i)$ is defined in Equation (3.31),

$$w_j^{(m)}(x_i) = m \cdot \left[w_j^{(1)}(x_i) w_i^{(m-1)}(x_i) - \frac{w_j^{(m-1)}(x_i)}{x_i - x_j} \right] \qquad \text{for} \quad i \neq j, \tag{3.33a}$$

$$w_j^{(m)}(x_i) = -\sum_{\substack{j=0 \\ j\neq i}}^{n} w_j^{(m)}(x_i) \qquad \text{for} \quad i = j. \tag{3.33b}$$

for $i, j = 0, 1, \ldots n$, and $m = 2, 3, \ldots, n-1$.

Here $w_j^{(m)}(x_i)$ is the mth derivative of the weight function.

3.2.4 Grid Spacing

A well-known problem associated with polynomial interpolation is that of oscillations at the edges of the interval These oscillations increase as the degree of the polynomial increases, and this issue is called Runge's phenomena There are two primary ways to cope with Runge's phenomena the first technique is to lower the degree of the polynomial by using several piecewise polynomials rather than one high degree polynomial, the second technique is to carefully select the abscissas such that they minimize these oscillations It has been shown that choosing equally spaced abscissas is not optimal for minimizing Runge's phenomena [8] Instead one should carefully choose grid locations that are typically *not* equally spaced, and these locations originate from various orthogonal polynomials In this work three different grid spacings are considered equally spaced abscissas, the roots of the Chebyshev polynomial of the first kind (also called the Chebyshev nodes), and the Gauss-Lobatto abscissas The remainder of this section is a brief explanation of how the latter two abscissas are determined

The Chebyshev nodes are the roots of the Chebyshev polynomials of the first kind ($\mathcal{T}$), which are defined over the interval $[-1, 1]$ The Chebyshev polynomial is never explicitly formed, instead one may directly compute the desired number of roots using a convenient formula show below For completeness the polynomial is presented here, and may be constructed using the recurrence relation shown below [3]

$$\mathcal{T}_0(x) = 1 \qquad \text{for} \quad n = 0,$$

$$\mathcal{T}_1(x) = x, \qquad \text{for} \quad n = 1,$$

$$\mathcal{T}_{n+1}(x) = 2\mathcal{T}_n(x) - T_{n-1}(x) \qquad \text{for} \quad n \geq 1$$

It has been shown that the nth Chebyshev polynomial has roots at

$$x = \cos\left(\frac{\pi(k - \frac{1}{2})}{n}\right) \qquad \text{for} \quad k = 1\ 2 \qquad n \tag{3 34}$$

Figure 3.4 shows $\mathcal{T}_{13}$ plotted over the interval $[-1, 1]$, and to emphasize that these nodes are not equally spaced, equally spaced grid locations have been marked. The roots (circled) are the desired abscissas to use in the polynomial interpolation.

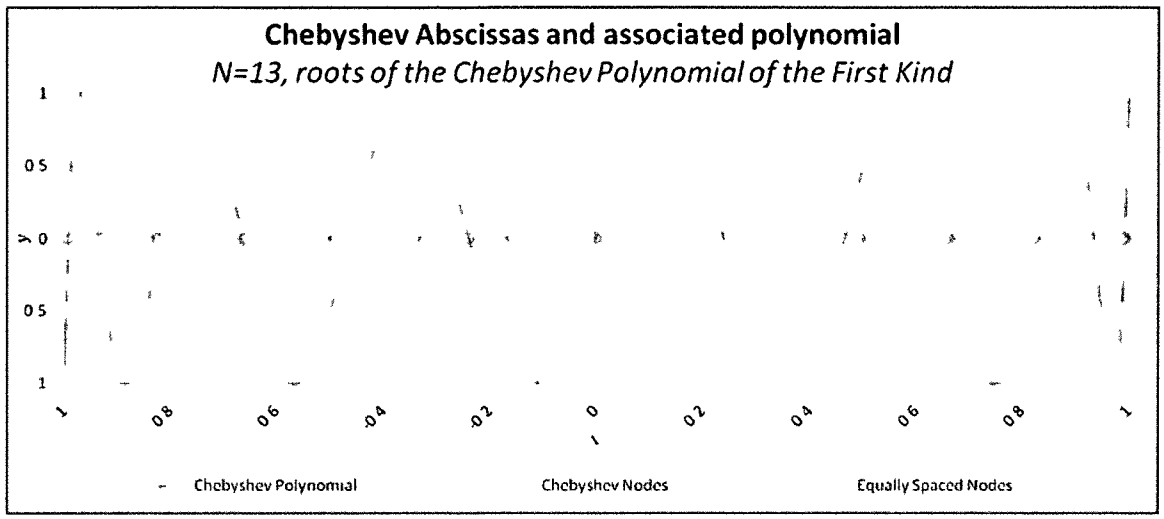


Figure 3.4: A Chebyshev polynomial of the first kind plotted over the interval [-1,1], with equally spaced grid locations marked.

The next grid spacing used called the Gauss-Lobatto nodes, are a variation of the standard Gaussian quadrature nodes, which are the roots of the nth degree Legendre polynomial $\mathcal{P}_n$. Gauss-Lobatto nodes differ from the standard Gaussian quadrature nodes because they use the once differentiated Legendre polynomial $\mathcal{P}'_{n-1}$, which yields $n-2$ roots. The remaining two abscissas are defined to be the endpoints of the interval -1 and 1. This distinguishes the Gauss-Lobatto nodes from many other orthogonal polynomial roots, including the Chebyshev nodes. Careful inspection of Figure 3.4 reveals that should one choose the Chebyshev nodes as the abscissas for interpolation, then one would not be able to obtain values at the boundaries. Figure 3.5 illustrates the differentiated Legendre polynomial $\mathcal{P}'_{12}$ and associated roots.

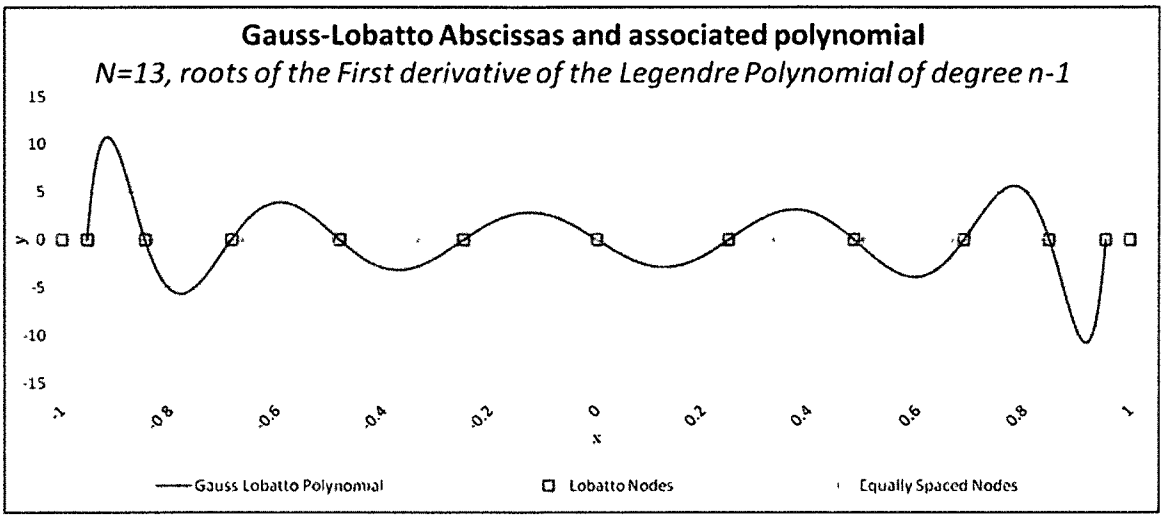


Figure 3.5: $\mathcal{P}'_{12}$ plotted over the interval [-1,1], with equally spaced grid locations marked.

3.2.5 High-order Derivatives via Lagrange Interpolating Polynomials

Having explained the fundamentals of differentiating the Lagrange interpolating polynomials as well as presenting three competing nodal selections, one must utilize these methods to compute high-order derivatives. Presented here are two different schemes for achieving this. The first analytically differentiates the Lagrange weights to the desired order, and the second treats the differentiated Lagrange weights as a differential operator, requiring that the weights only be differentiated once.

The first scheme follows Shu's method [27] and uses Equation (3.33) to analytically differentiate the weights. To obtain an mth-order derivative one would follow the scheme described below

$$f'(x_i) = \sum_{j=0}^{n} w'_j(x_i) f(x_j), \tag{3.35a}$$

$$f''(x_i) = \sum_{j=0}^{n} w_j''(x_i) f(x_j), \tag{3.35b}$$

$$\vdots$$

$$f^{(m)}(x_i) = \sum_{j=0}^{n} w_j^{(m)}(x_i) f(x_j). \tag{3.35c}$$

The second scheme is proposed in this dissertation for the first time. We have noted from Equation (3.31), that the weight function is computed independent of the function values of $f(x)$ and depends solely upon the location on the grid. In this sense, $w_j(x)$ differentiates some function values f at these particular points, and may be treated as a differentiation operator. Recognizing this, the following scheme is proposed based on the first derivative weight function

$$f'(x_i) = \sum_{j=0}^{n} w_j'(x_i) f(x_j), \tag{3.36a}$$

$$f''(x_i) = \sum_{j=0}^{n} w_j'(x_i) f'(x_j), \tag{3.36b}$$

$$\vdots$$

$$f^{(m)}(x_i) = \sum_{j=0}^{n} w_j'(x_i) f^{(m-1)}(x_j). \tag{3.36c}$$

The following is a comparison of results when using each of the schemes listed above, but before presenting results, we will discuss the motivation for seeking an alternative to differentiating the weights multiple times. The computational algorithm associated with Equation (3.31) is shown in Algorithm 3.1. Similarly, the algorithm associated with either Equation (3.36a) or Equation (3.35a) is shown in Algorithm 3.2. Regardless of the scheme used, these algorithms must be used at least once. Particularly, Algorithm 3.2 must be computed with either successively differentiated weights $w[i][j]$ (Shu's method), or with successively differentiated function values (our proposed method).

Algorithm 3.1: Computational algorithm used to obtain the first derivative weights, based on the analytical form in Equation (3.31).

Input: Set of N abscissas
Output: Differentiated weight function $w_j(x_i)$ evaluated at each x_i

```
for i = 1 to N do
    for j = 1 to N do
        if i ≠ j then
            num = 1.0
            den = 1.0
            for k = 1 to N do
                if k ≠ i and k ≠ j then
                    num = num · (i − k)
                    den = den · (j − k)
                end
            end
            w[i][j] = num/(den · (x[j] − x[i]))
        end
    end
end
for i = 1 to N do
    w[i][j] = 0.0
    for k = 1 to N do
        if k ≠ i then
            w[i][j] = w[i][j] + 1.0/x[j] − x[i])
        end
    end
end
return w
```

To obtain a solution using the weights requires evaluating the differentiated Lagrange interpolating polynomial at the grid locations x_j, for $j = 1, \ldots, N$. A portion of the polynomial evaluation is handled when computing the weights, but one must still compute the linear combination of the weight with each function value to determine the differentiated function value $f'(x_i)$. This computation is greatly simplified since the weights are already computed.

Algorithm 3.2: Computational algorithm used to obtain the differentiated function values using the differentiated Lagrange weights.

Input: Set of N abscissas
Input: 2D array of size $N \times N$ containing evaluated Lagrange weights
Output: The function f differentiated at each abscissa
for $i = 1$ **to** N **do**
  $f1[i] = 0.0$
  for $j = 1$ **to** N **do**
    $f1[i] = f1[i] + w[i][j] \cdot f[j]$
  end
end
return $f1$

From the algorithms shown in Algorithm 3.1 and Algorithm 3.2 one may note that the algorithm to obtain the first derivative weights (Algorithm 3.1) has $O(N^3)$ computational complexity, and the computation required to evaluate the derivatives (Algorithm 3.2) has computational complexity $O(N^2)$. It can be shown that Shu's method (Equation (3.33)) requires $O(mN^2)$ computational complexity, where m is the order of the derivative desired and $m > 1$. The computational complexity to evalute an mth derivative using Shu's method is then

$$O\Big(\underbrace{N^3}_{w_j'(x_i) \text{ calculation}} + \overbrace{mN^2}^{w_j^{(m)}(x_i) \text{ calculation}} + \underbrace{mN^2}_{f^{(m)}(x_i) \text{ calculation}} \Big) = O(N^3 + 2mN^2), \quad (3.37)$$

and the computational complexity to evaluate an mth derivative using the proposed method is

$$O\Big(\underbrace{N^3}_{w_j'(x_i) \text{ calculation}} + \underbrace{mN^2}_{f^{(m)}(x_i) \text{ calculation}} \Big) = O(N^3 + mN^2) \quad (3.38)$$

The spatial complexity, which is a bound on the storage requirement, is another metric that should be noted. For Shu's method shown in Equation (3.33). the first derivative weights must be stored. as well as the previous $(m-1)$th derivative weights to compute the mth derivative weights. In terms of storage, a

set of differentiated weights is a two-dimensional array requiring $N \times N$ storage, and three of these must be retained at any give time This translates to a spatial complexity of

$$O(3N^2) \tag{3 39}$$

The proposed scheme only requires that the first derivative weights be retained, and so the spatial complexity is

$$O(N^2) \tag{3 40}$$

In summary, analytically differentiating the Lagrange weights up to the mth derivative requires $O(N^3 + 2mN^2)$ computation and $O(3N^2)$ storage, whereas treating the differentiated Lagrange weights as an operator requires $O(N^3 + mN^2)$ computation and $O(N^2)$ storage Between the two methods it is clear that one must always have the first derivative weights and one must always use these weights to obtain the differentiated function values making the minimum spatial complexity $O(N^2)$ and the minimum computational complexity $O(N^3 + mN^2)$ for $m > 1$ With the minimal requirements in mind, Shu's algorithm then requires an additional $O(mN^2)$ amount of computation and $O(2N^2)$ amount of storage, while the proposed scheme requires only the minimal amount of computation and storage, $O(N^3+mN^2)$ and $O(N^2)$ respectively

A comparison of solutions using both Shu s method and the scheme proposed above are presented below The function

$$f(x) = e^x \tag{3 41}$$

was chosen as a test function and the first- through sixth-order derivatives were computed using each method

From Table 3 1 one can see that both methods produce nearly identical results with the error on the same order of magnitude In the above table the error is

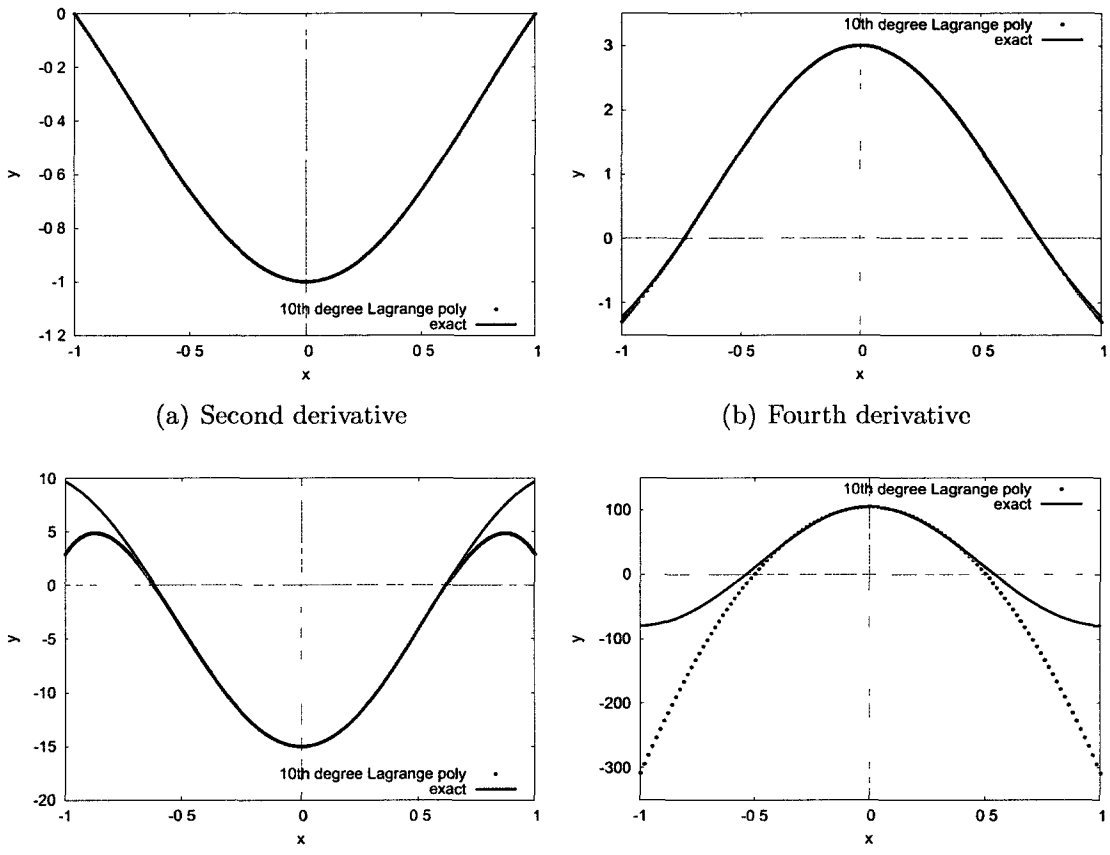
computed by analytically differentiating Equation (3 41), and then computing the absolute error To summarize the error across all interpolated values, the L_2 norm is used as well as the infinity norm L_∞ which in this case is the maximum error

Table 3 1 Comparison of derivatives obtained via Shu's method and those from the proposed method

			Shu		Proposed	
Eq	N	m	L_2	L_∞	L_2	L_∞
(3 41)	11	2	$3\,99 \times 10^{-05}$	$3\,98 \times 10^{-05}$	$2\,19 \times 10^{-05}$	$2\,14 \times 10^{-05}$
	11	4	$7\,72 \times 10^{-02}$	$6\,89 \times 10^{-02}$	$4\,07 \times 10^{02}$	$3\,68 \times 10^{-02}$
	11	6	$8\,14 \times 10^{+00}$	$7\,11 \times 10^{+00}$	$2\,05 \times 10^{+00}$	$1\,22 \times 10^{+00}$
	11	8	$1\,43 \times 10^{+03}$	$1\,36 \times 10^{+03}$	$7\,79 \times 10^{+02}$	$6\,96 \times 10^{+02}$

3.2.6 Abscissa Impact on Differentiation Error

Having shown how to compute high-order derivatives, results will now be presented using the grid spacings highlighted in Section 3 2 4 To evaluate the various spacings, a test function was chosen and analytically differentiated, and then the solution was approximated using a differentiated Lagrange interpolating polynomial The followings figures will vary based on the grid spacing used to construct the Lagrange interpolating polynomial The first grid spacing shown in Figure 3 6 uses equally spaced abscissas



(a) Second derivative

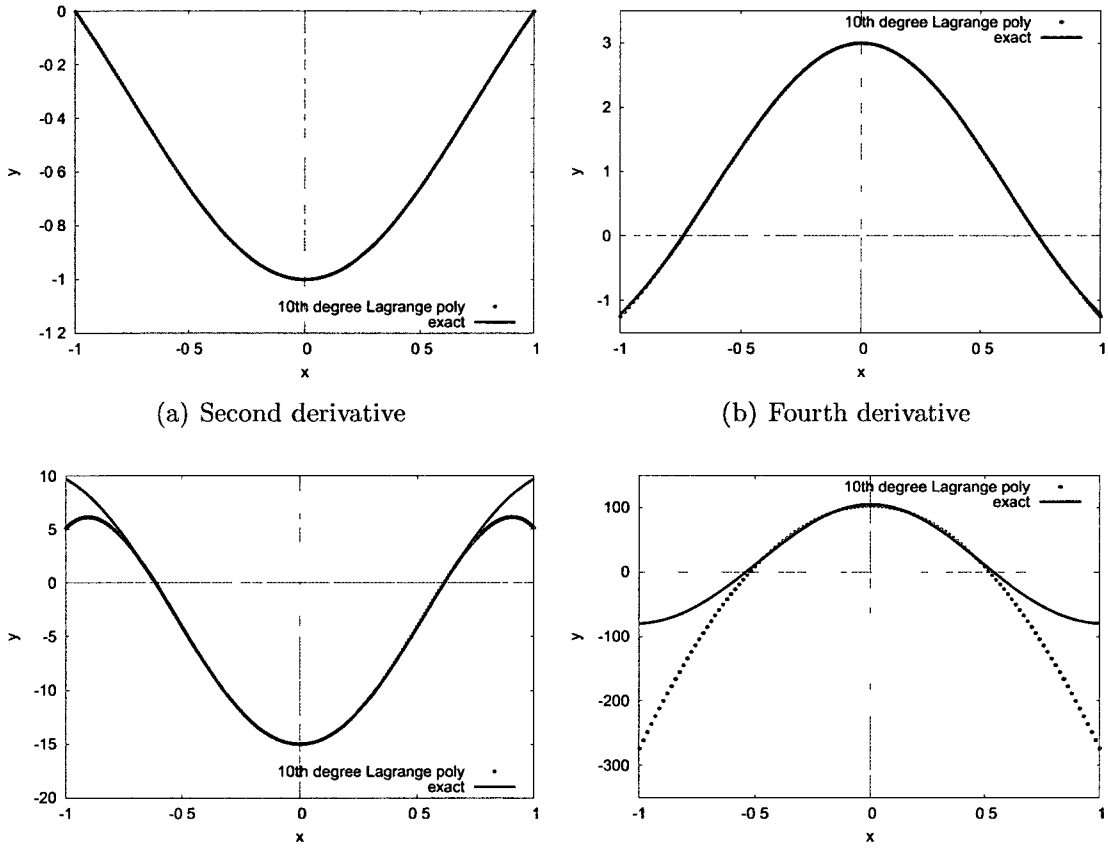
(b) Fourth derivative

(c) Sixth derivative

(d) Eighth derivative

Figure 3.6: Differentiated tenth degree Lagrange polynomial for the function $f(x) = e^{(-\frac{x^2}{2})}$, over the interval $[-1, 1]$ using equally spaced abscissas, as well as the exact solution.

One can see that the accuracy of the differentiated Lagrange interpolating polynomial slowly degrades as successive differentiation is performed. Specifically, Figure 3.6(d), which contains the plot of the eighth derivative, shows considerable error throughout most of the domain. One can visually see the error beginning to appear at the endpoints of the fourth derivative plot in Figure 3.6(b) as well. In Figure 3.7 the abscissas have been changed to the Chebyshev nodes, and in Figure 3.8 the Gauss-Lobatta nodes have been used.



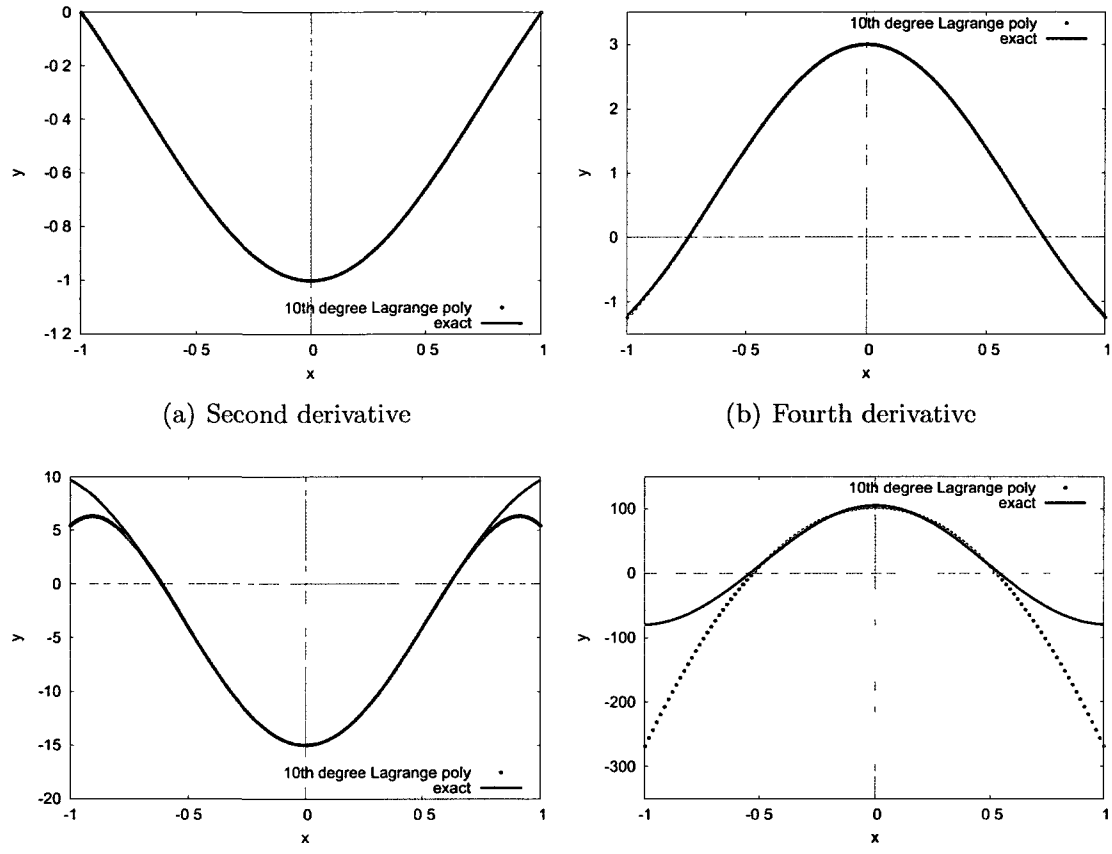
(a) Second derivative

(b) Fourth derivative

(c) Sixth derivative

(d) Eighth derivative

Figure 3.7: Differentiated tenth degree Lagrange polynomials for the function $f(x) = e^{(-\frac{x^2}{2})}$, over the interval $[-1, 1]$, using the Chebyshev nodes as abscissas.



(a) Second derivative

(b) Fourth derivative

(c) Sixth derivative

(d) Eighth derivative

Figure 3.8: Differentiated tenth degree Lagrange polynomials for the function $f(x) = e^{(-\frac{x^2}{2})}$, over the interval $[-1, 1]$, using the Gauss-Lobatto nodes as abscissas.

While these plots are presented primarily to aid in visualizing what happens as Lagrange interpolating polynomials are successively differentiated, one can still graphically see the impact of grid spacing. Observe Figures 3.6(d), 3.7(d), and 3.8(d), one can clearly see that in Figure 3.6(d) the endpoints of the approximation are yielding a solution beyond -300, while the approximations using orthogonal polynomials still have considerable error, but it is noticeably smaller than that of the error in the equidistant grid spacing. Which leads to the next section, where instead of plotting the differentiated functions, instead the $\log_{10}$ of the error is plotted, allowing one to view the distribution of the error and directly compare the central differences to the differentiated Lagrange interpolating polynomials.

3.3 Method Comparison

In this section results will be presented using the techniques presented throughout this chapter. Four functions have been selected for use as test functions, and comparisons will take place on two different intervals. The following numerical differentiation techniques will be employed:

- Second-order accurate central difference approximation of the Laplace operator,
- Fourth-order accurate central difference approximation of the Laplace operator,
- Sixth-order accurate central difference approximation of the Laplace operator,
- Twelfth degree piecewise differentiated Lagrange interpolating polynomial with equally spaced abscissas,
- Twelfth degree piecewise differentiated Lagrange interpolating polynomial with the Chebyshev nodes as abscissas,
- Twelfth degree piecewise differentiated Lagrange interpolating polynomial with the Gauss-Lobatto nodes as abscissas.

3.3.1 Graph Interpretation

Before approaching the comparisons, one must understand the format the data is presented in. For example, the function $f(x) = e^{(-\frac{x^2}{2})}$ will be used to demonstrate what to look for in the plots. Figure 3.9 shows the test function plotted over the interval $[0, 1.035]$. The test function is now analytically differentiated, as well as differentiated using a once differentiated tenth degree Lagrange interpolating polynomial using equally spaced abscissas.

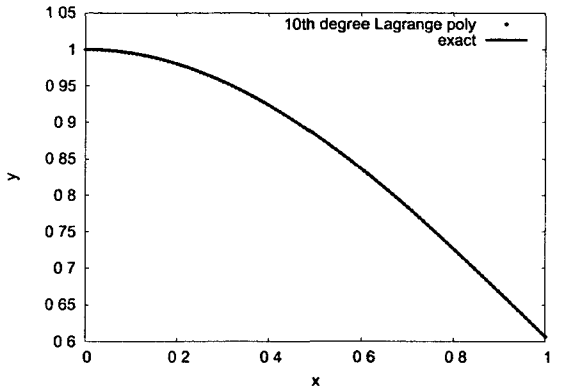
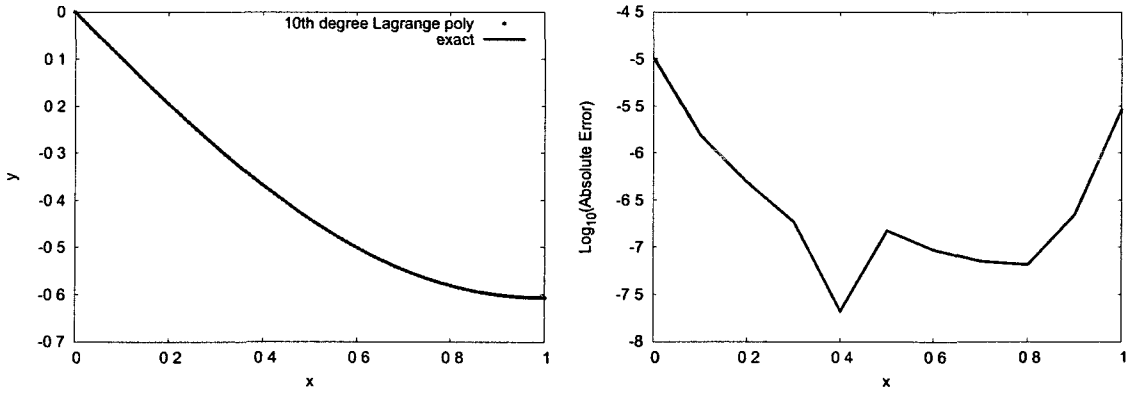


Figure 3.9: Test function $f(x) = e^{(-\frac{x^2}{2})}$ plotted over the interval $[0, 1.035]$.

Figure 3.10(a) shows the analytical solution in black, and the Lagrange approximation in red. From Figure 3.10(a) it is unclear how much the approximation varies from the actual solution, and so the absolute error is computed, and the $\log_{10}$ of this error is plotted in Figure 3.10(b). This error plot shows a significant crux of the differentiated Lagrange interpolated polynomials, and that is the error near the endpoints may be considerably larger than that of the error in the center of the interval.

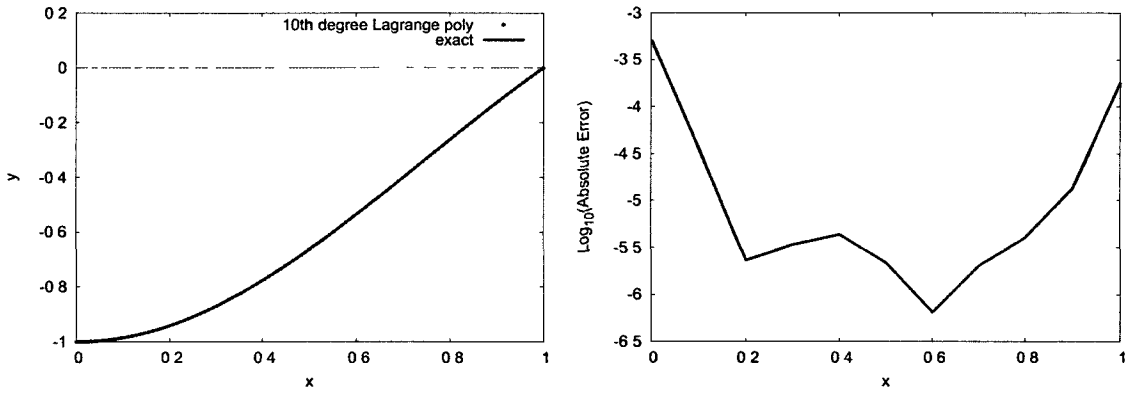


(a) First derivative

(b) Absolute error of the first derivative

Figure 3.10: First derivative of the test function $f(x) = e^{(-\frac{x^2}{2})}$ and associated absolute error over the interval $[0, 1.035]$.

Next, the test function is differentiated twice in Figure 3.11(a), and the error is shown in Figure 3.11(b). One should note from the error plot, that the error has shifted by nearly two orders of magnitude (-5 to -3.25). And as in the plots of the first derivative error one can see the "U" shape of the absolute error curve. The test function is now differentiated up to the sixth derivative. To view the complete set of first-order through sixth-order derivatives, the reader is directed to Appendix A.

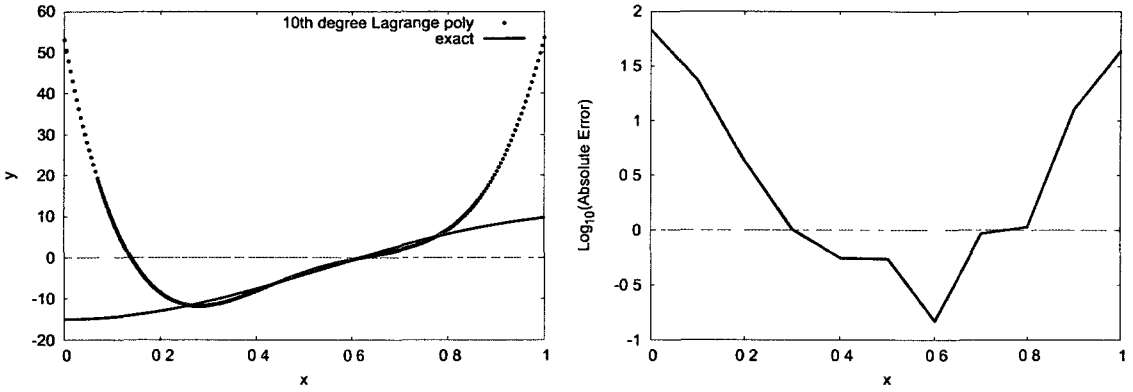


(a) Second derivative

(b) Absolute error of the second derivative

Figure 3.11: Second derivative of the test function $f(x) = e^{(-\frac{x^2}{2})}$ and associated absolute error over the interval $[0, 1.035]$.

From Figure 3.12(a) note the degree to which the approximation varies from analytical, with the absolute error in Figure 3.12(b) showing the order of magnitude of the error in the range -1 to nearly 2, which is to say, the approximation of the sixth derivative is off by nearly 1×10^2. To cope with this, piecewise Lagrange polynomials are used, which does lower the error slightly, and in the curves showing the actual tests, one should note the "hills" and "valleys", as these "U" shapes indicate a piecewise polynomial.



(a) Sixth derivative

(b) Absolute error of the sixth derivative

Figure 3.12: Sixth derivative of the test function $f(x) = e^{(-\frac{x^2}{2})}$ and associated absolute error over the interval $[0, 1.035]$.

As a final example, Figure 3.13 shows a typical error plot. The function tested in this plot is show in Figure 3.13(a), and is the same test function used in the previous examples. The plot shows the $\log_{10}$ of the absolute error obtained by approximating the first- through sixth-order derivatives using a differentiated twelfth degree piecewise Lagrange interpolating polynomial. One may identify the derivatives approximated using the legend highlighted by Figure 3.13(c). It should be noted that the Lagrange error plots will contain all derivatives from the first to the sixth, while central difference error plots will contain only the even-order derivatives, this is because the central differences approximated the Laplace operator. The procedure outlined in Equation (3.36), was used to compute each derivative using differentiated twelfth degree piecewise Lagrange interpolating polynomials. In this particular case, one may see from Figure 3.13(b) the abscissas used were equally spaced, and in other Lagrange error plots this may be either equally spaced, Chebyshev nodes, or Gauss-Lobatto nodes.

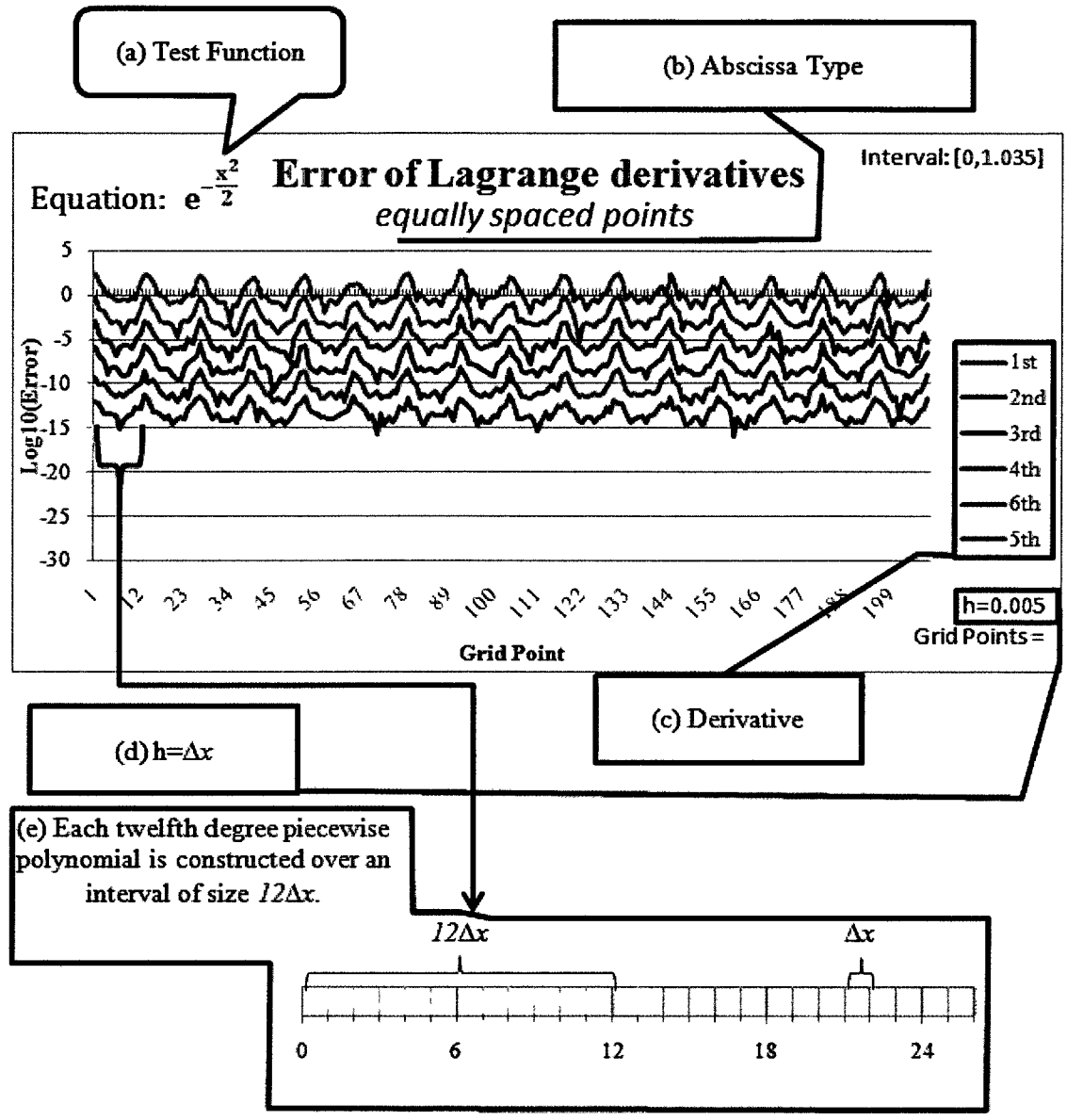


Figure 3.13: Differentiated piecewise thirteenth degree Lagrange polynomials for the function $f(x) = e^{(-\frac{x^2}{2})}$, over the interval $[0, 1.035]$, utilizing 208 total grid points.

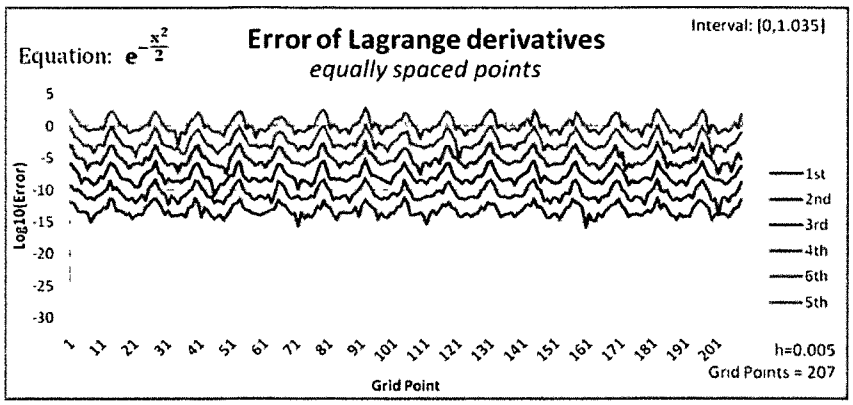
Additional attention must be paid to Figure 3.13(e), because piecewise polynomials are used, it must be understood how the large interval is partitioned into the smaller intervals and how Δx is relevant to this partitioning. To partition the large interval the desired total number of grid points is fixed to be a multiple of thirteen, and the domain is partitioned into regions of size $12\Delta x$. Supposing Chebyshev nodes were desired, thirteen Chebyshev nodes would be constructed

throughout this subinterval This division guarantees that regardless of the abscissa selection within the piecewise interval, that the entire domain $[a, b]$ is interpolated avoiding having points clustered at the endpoints of the large interval In example, given 208 points, each subinterval will contain 13 abscissas within that specific region, be it Chebyshev roots, Gauss-Lobatto nodes, or simply using the equally spaced points This is distinctly different from taking the interval $[a, b]$ and creating 208 abscissas of a particular type and then creating a interpolating polynomial only though 13 points at a time The difference being that creating 208 abscissas across the entire interval, would not necessarily create optimal nodes to minimize Runge's phenomena should you only use 13 points to create the interpolation Should you create 208 abscissas across the entire interval, those abscissas would be designed to minimize Runge's phenomena for a 207th degree polynomial

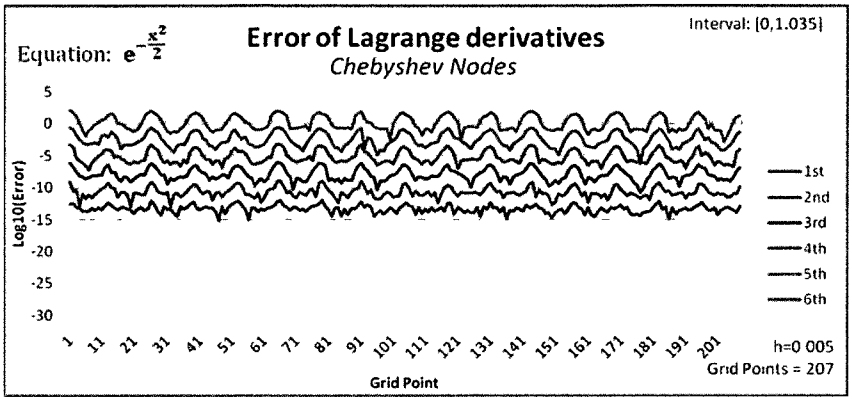
3.3.2 Error Plots

On the following pages error plots are shown for the test function $f(x) = e^{(-\frac{x^2}{2})}$ over two intervals $[0, 1.035]$ and $[0, 10.35]$ using both the Lagrange and central difference differentiation techniques described previously to compute the first- through sixth-order derivatives in the case of Lagrange differentiation and second, fourth, and sixth-order derivatives in the case of central differences approximations. Figure 3.14 shows three plots that use differentiated piecewise twelfth degree Lagrange interpolating polynomials, with Figure 3.14(a) using equally spaced abscissas, Figure 3.14(b) using the Chebyshev nodes, and Figure 3.14(c) using the Gauss-Lobatto nodes. Figure 3.15 on the following page shows the error over the same interval but using the central difference approximation of the Laplace operator, with Figure 3.15(a) using a second-order accurate central difference, Figure 3.15(b) using a fourth-order accurate central difference, and Figure 3.15(c) using a sixth-order accurate central difference.

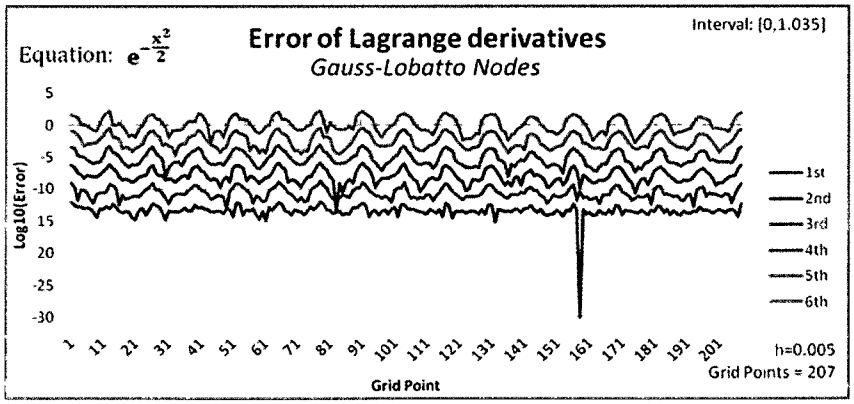
The follow pages also contain the same test function differentiated over a larger interval, while maintaining the same number of grid points, which implies that Δx has become larger. Additional test functions are plotted in a similar manner and may be viewed in Appendix B. In Section 3.4, the data from the test function shown here will be analyzed. The observation that should be made from the Figures presented both here and in the Appendix, is that the fourth and sixth-order accurate central difference approximations are roughly as accurate the piecewise twelfth degree Lagrange interpolating polynomial approximations The notable difference between the Finite Difference approximations and Lagrange approximations are the oscillations in the error of the Lagrange approximations.



(a) Differentiated Lagrange interpolating polynomials using equally spaced nodes to compute the first- through sixth-order derivatives.

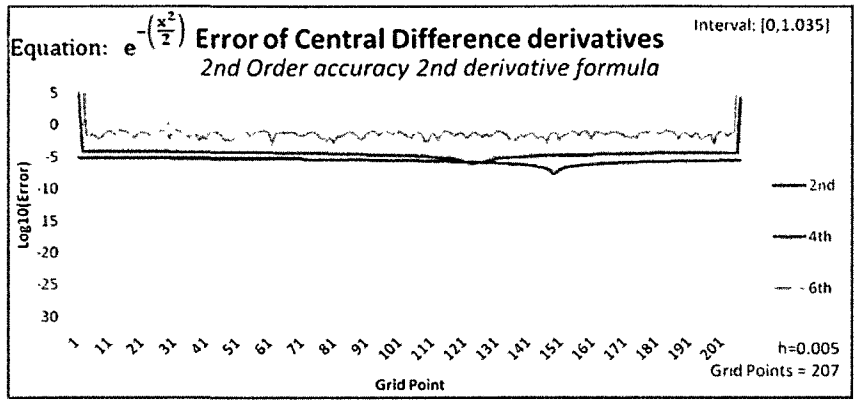


(b) Differentiated Lagrange interpolating polynomials using the Chebyshev nodes to compute the first- through sixth-order derivatives.

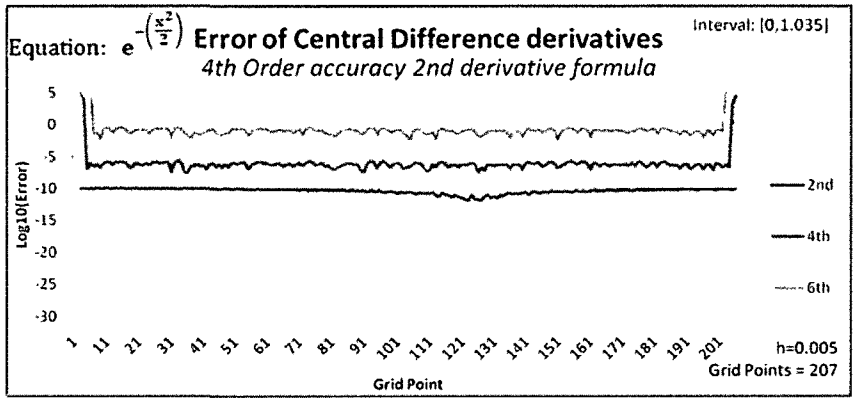


(c) Differentiated Lagrange interpolating polynomials using the Gauss-Lobatto nodes to compute the first- through sixth-order derivatives.

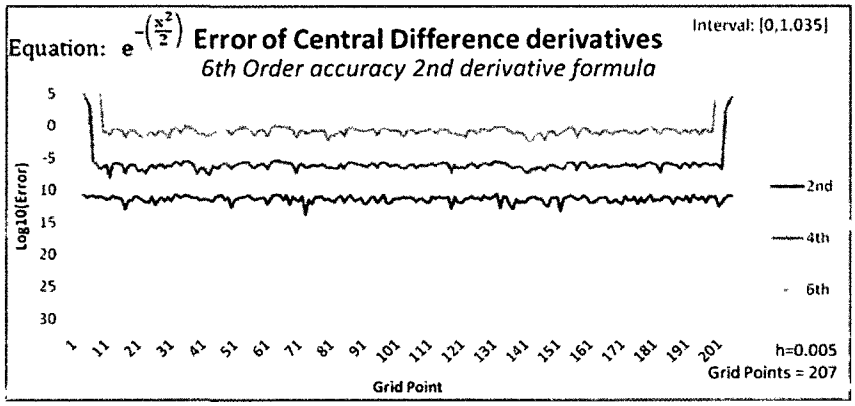
Figure 3.14: Differentiated piecewise twelfth degree Lagrange interpolating polynomials for the function $f(x) = e^{(-\frac{x^2}{2})}$, over the interval $[0, 1.035]$, utilizing 208 total grid points.



(a) Second-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

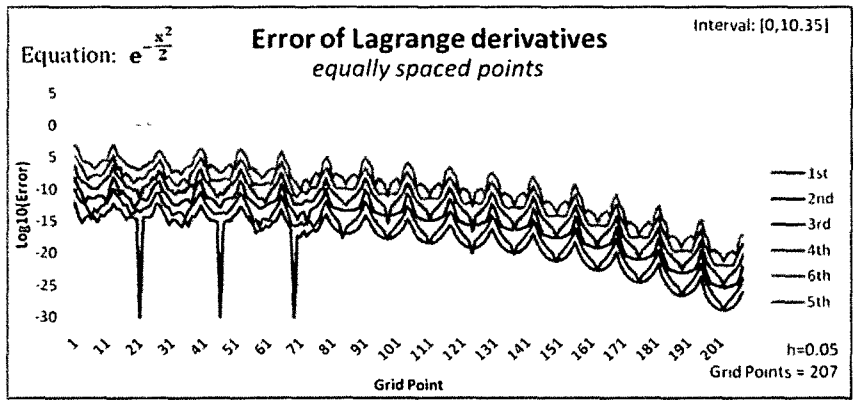


(b) Fourth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

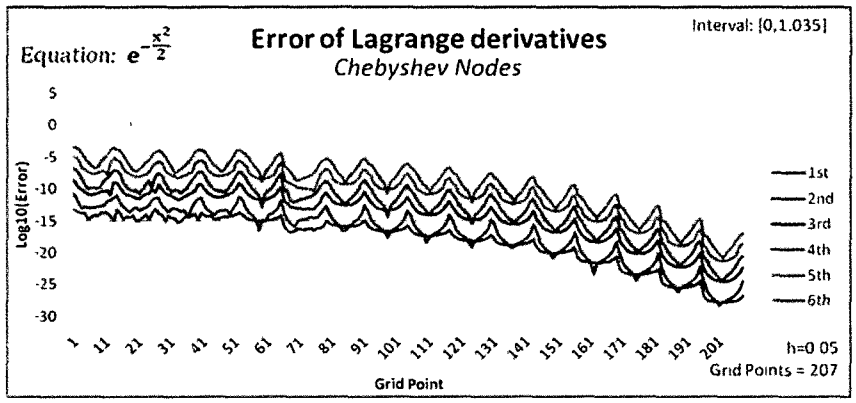


(c) Sixth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

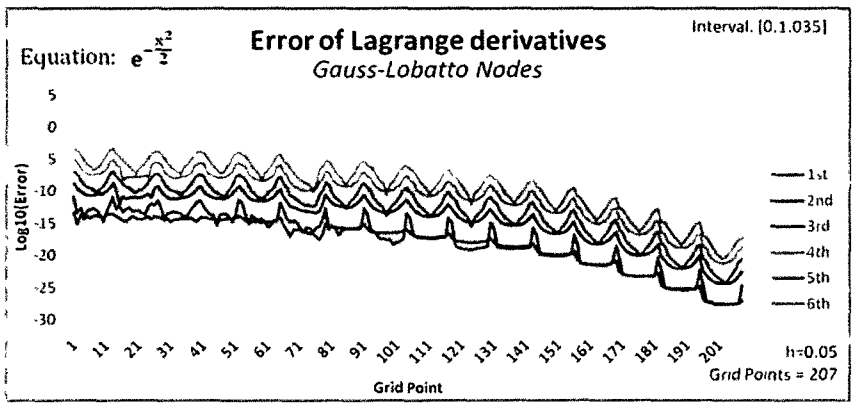
Figure 3.15: Central difference approximations of the Laplace operator applied to the function $f(x) = e^{(-\frac{x^2}{2})}$, over the interval $[0, 1.035]$, utilizing 208 total grid points, and various orders of accuracy.



(a) Differentiated Lagrange interpolating polynomials using equally spaced nodes to compute the first- through sixth-order derivatives.

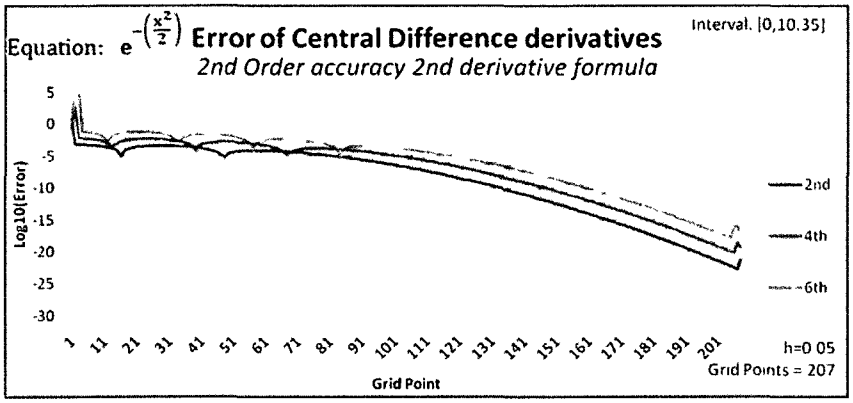


(b) Differentiated Lagrange interpolating polynomials using the Chebyshev nodes to compute the first- through sixth-order derivatives.

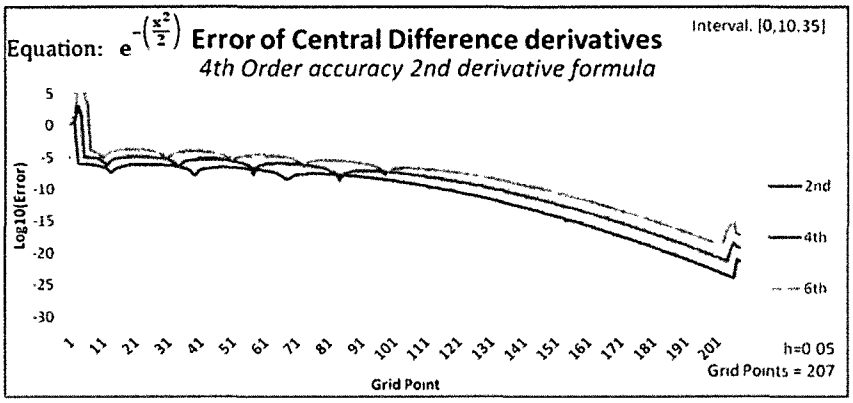


(c) Differentiated Lagrange interpolating polynomials using the Gauss-Lobatto nodes to compute the first- through sixth-order derivatives.

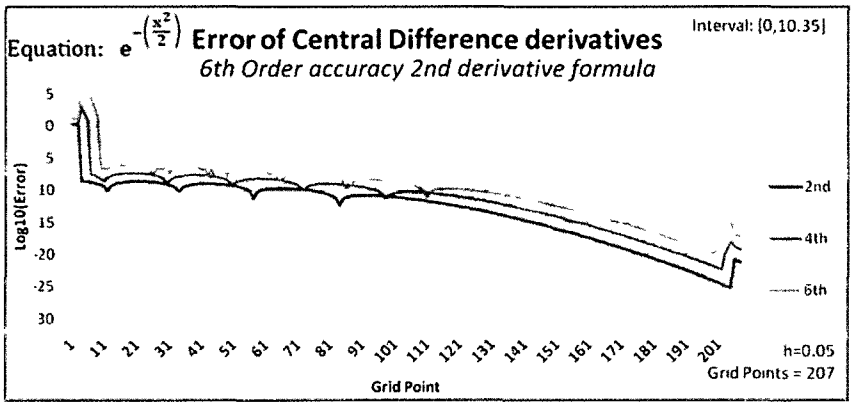
Figure 3.16: Differentiated piecewise twelfth degree Lagrange interpolating polynomials for the function $f(x) = e^{(-\frac{x^2}{2})}$, over the interval $[0, 10.35]$, utilizing 208 total grid points.



(a) Second-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.



(b) Fourth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.



(c) Sixth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

Figure 3.17: Central difference approximations of the Laplace operator applied to the function $f(x) = e^{(-\frac{x^2}{2})}$, over the interval $[0, 10.35]$, utilizing 208 total grid points, and various orders of accuracy.

3.4 Conclusions

The preceding error plots, as well as those in Appendix B provide a high level view of the behavior of the Finite Difference and Lagrange differentiation techniques discussed in this chapter To objectively compare these methods two common metrics have been used, the root mean square and infinity norm The root mean square RMS defined as

$$RMS = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(Err_i)^2}, \tag{3.42}$$

where Err_i is the absolute error at grid point $i = 0, \quad , n$ In this context the RMS provides an estimate of what the observed error is The infinity norm L_∞ is the maximum error observed

$$L_\infty = \max(|Err_0| \qquad |Err_n|) \tag{3.43}$$

Several tables are presented showing the above error metrics computed for the test function $f(x) = e^{(-\frac{x^2}{2})}$ over the interval $[0, 1\,035]$, utilizing 208 total grid points, and $\Delta x = 0\,005$ To interpret the tables the following list describes what the column heading *Method* indicates

- *Equidistant* differentiated piecewise twelfth degree Lagrange interpolating polynomial using equally spaced abscissas,
- *Chebyshev* differentiated piecewise twelfth degree Lagrange interpolating polynomial using the Chebyshev nodes
- *Gauss Lobatto* differentiated piecewise twelfth degree Lagrange interpolating polynomial using the Gauss-Lobatto nodes,
- $O(\Delta x^2)$ second-order accurate central difference approximation of the Laplace operator,
- $O(\Delta x^4)$ fourth order accurate central difference approximation of the Laplace operator,
- $O(\Delta x^6)$ sixth order accurate central difference approximation of the Laplace operator

Table 3 2 shows the above error metrics for the first through fourth derivatives One should note that for the odd-order derivatives, the central difference approximations are not shown, and the solid line indicates these methods were uncomputable

Table 3 2 Root Mean Square and maximum absolute error for the first through fourth-order derivatives of the test function $f(x) = e^{(-\frac{x^2}{2})}$ with $\Delta x = 0\,005$

Error for the Derivatives of the Test Function $f(x) = e^{(\frac{x^2}{2})}$			
Deriv	Method	RMS	L_∞
1st	Equidistant	$7\,0947 \times 10^{14}$	$5\,8616 \times 10^{-12}$
	Chebyshev	$1\,9348 \times 10^{-13}$	$1\,0560 \times 10^{12}$
	Gauss-Lobatto	$1\,8662 \times 10^{13}$	$1\,0570 \times 10^{-12}$
	$O(\Delta x^2)$	———	———
	$O(\Delta x^4)$	———	———
	$O(\Delta x^6)$	———	———
2nd	Equidistant	$7\,1019 \times 10^{-11}$	$6\,6711 \times 10^{-09}$
	Chebyshev	$2\,0596 \times 10^{-10}$	$1\,0126 \times 10^{09}$
	Gauss-Lobatto	$2\,5238 \times 10^{10}$	$1\,6979 \times 10^{09}$
	$O(\Delta x^2)$	$3\,7798 \times 10^{06}$	$6\,2496 \times 10^{-06}$
	$O(\Delta x^4)$	$6\,3121 \times 10^{-11}$	$1\,1303 \times 10^{10}$
	$O(\Delta x^6)$	$7\,5022 \times 10^{12}$	$2\,1610 \times 10^{11}$
3rd	Equidistant	$5\,4653 \times 10^{08}$	$5\,9982 \times 10^{06}$
	Chebyshev	$1\,8355 \times 10^{07}$	$9\,2268 \times 10^{-07}$
	Gauss-Lobatto	$2\,3015 \times 10^{07}$	$1\,4963 \times 10^{06}$
	$O(\Delta x^2)$	———	———
	$O(\Delta x^4)$	———	———
	$O(\Delta x^6)$	———	———
4th	Equidistant	$3\,1768 \times 10^{05}$	$3\,6396 \times 10^{03}$
	Chebyshev	$1\,2313 \times 10^{04}$	$5\,8485 \times 10^{04}$
	Gauss-Lobatto	$1\,4809 \times 10^{04}$	$8\,9775 \times 10^{04}$
	$O(\Delta x^2)$	$5\,7761 \times 10^{+03}$	$8\,0000 \times 10^{+04}$
	$O(\Delta x^4)$	$7\,2801 \times 10^{+03}$	$9\,9997 \times 10^{+04}$
	$O(\Delta x^6)$	$7\,9992 \times 10^{+03}$	$1\,0888 \times 10^{+05}$

In the following Table 3 3 as well as in Table 3 2 one will notice that the error metrics for the central difference approximations tend to become high The reason for this is clear from the plots as the uncomputable end points of the central difference

approximations become extremely inaccurate as the order of the derivative increases, and these significantly inaccurate points dominate the error metrics. This is not desirable, as the error across the majority of the domain for the central difference approximations is accurate, with only 5-10 points skewing the metrics.

Table 3.3: Root Mean Square and maximum absolute error for the fifth-order and sixth-order derivatives of the test function $f(x) = e^{(-\frac{x^2}{2})}$ with $\Delta x = 0.005$.

Error for the Derivatives of the Test Function $f(x) = e^{(-\frac{x^2}{2})}$			
Deriv.	Method	RMS	L_∞
5th	Equidistant	1.4623×10^{-02}	$1.6825 \times 10^{+00}$
	Chebyshev	6.4160×10^{-02}	2.9421×10^{-01}
	Gauss-Lobatto	7.2778×10^{-02}	4.0557×10^{-01}
	$O(\Delta x^2)$	———	———
	$O(\Delta x^4)$	———	———
	$O(\Delta x^6)$	———	———
6th	Equidistant	$5.4594 \times 10^{+00}$	$6.1670 \times 10^{+02}$
	Chebyshev	$2.6534 \times 10^{+01}$	$1.2029 \times 10^{+02}$
	Gauss-Lobatto	$2.8420 \times 10^{+01}$	$1.4385 \times 10^{+02}$
	$O(\Delta x^2)$	$5.1663 \times 10^{+08}$	$6.4001 \times 10^{+09}$
	$O(\Delta x^4)$	$8.7081 \times 10^{+08}$	$1.0355 \times 10^{+10}$
	$O(\Delta x^6)$	$1.0834 \times 10^{+09}$	$1.2528 \times 10^{+10}$

To combat the extremely large error observed near the end points of the central difference approximations, the following metrics were computed ignoring the first and last ten points of each approximation. Table 3.4 shows the error metrics for the even-order derivatives using these shortened intervals. To emphasize the impact this has on the error metrics, the smallest RMS for each derivative has been bolded. From this table it becomes clearer that the central difference approximations may yield a more accurate solution than the Lagrange differentiation method if one is able to either approximate the solution at the uncomputable end points, or utilize a sufficient number of points that the solution may be discarded near the boundaries.

Table 3 4 Root Mean Square and maximum absolute error for the even-order derivatives of the test function $f(x) = e^{(-\frac{x^2}{2})}$ with $\Delta x = 0\,005$, after removing the first and last 10 points of the Central Difference approximations

Error for the Derivatives of the Test Function $f(x) = e^{(\frac{x^2}{2})}$			
Deriv	Method	RMS	L_∞
2nd	Equidistant	$7\,1019 \times 10^{11}$	$6\,6711 \times 10^{09}$
	Chebyshev	$2\,0596 \times 10^{-10}$	$1\,0126 \times 10^{-09}$
	Gauss-Lobatto	$2\,5238 \times 10^{-10}$	$1\,6979 \times 10^{-09}$
	$O(\Delta x^2)$	$3\,6571 \times 10^{-06}$	$6\,2028 \times 10^{-06}$
	$O(\Delta x^4)$	$5\,9858 \times 10^{-11}$	$1\,1303 \times 10^{-10}$
	$O(\Delta x^6)$	$\mathbf{7\,2745 \times 10^{-12}}$	$2\,1610 \times 10^{-11}$
4th	Equidistant	$3\,1768 \times 10^{05}$	$3\,6396 \times 10^{03}$
	Chebyshev	$1\,2313 \times 10^{-04}$	$5\,8485 \times 10^{-04}$
	Gauss-Lobatto	$1\,4809 \times 10^{-04}$	$8\,9775 \times 10^{-04}$
	$O(\Delta x^2)$	$3\,6006 \times 10^{-05}$	$6\,2428 \times 10^{-05}$
	$O(\Delta x^4)$	$\mathbf{8\,2059 \times 10^{-07}}$	$1\,9764 \times 10^{-06}$
	$O(\Delta x^6)$	$1\,1897 \times 10^{-06}$	$4\,2159 \times 10^{06}$
6th	Equidistant	$5\,4594 \times 10^{+00}$	$6\,1670 \times 10^{+02}$
	Chebyshev	$2\,6534 \times 10^{+01}$	$1\,2029 \times 10^{+02}$
	Gauss-Lobatto	$2\,8420 \times 10^{+01}$	$1\,4385 \times 10^{+02}$
	$O(\Delta x^2)$	$\mathbf{5\,8604 \times 10^{-02}}$	$1\,4722 \times 10^{01}$
	$O(\Delta x^4)$	$1\,4401 \times 10^{01}$	$3\,2562 \times 10^{-01}$
	$O(\Delta x^6)$	$2\,4755 \times 10^{01}$	$8\,9009 \times 10^{01}$

Based on the results presented here, it appears using differentiated Lagrange interpolating polynomials may provide accuracy near or in some cases better than that of the central difference approximations But the large oscillations in the error throughout the entire interval pose some challenges, and in the context of the FDTD-Q method the stability of the Lagrange method is questionable Several numerical experiments were performed using the above Lagrange differentiation scheme to compute derivatives for the Generalized FDTD-Q method in solving a model problem with exact solution but in all cases the Generalized FDTD-Q scheme became unstable and failed to converge to the solution It is also noted from Table 3 4 that the most accurate approximation of the sixth derivative was obtained via a second-order accurate method, while the sixth-order accurate scheme produced

results more accurate than the second-order accurate scheme for the second and fourth derivatives

Based on these findings, we propose future work in developing a hybrid Lagrange/Finite Difference method that uses a differentiated Lagrange interpolating polynomial to compute only the end points, which are then fed into a Finite Difference approximation Based on the need to prove what the stability condition of the resulting scheme is, the we have elected to utilize a sixth-order accurate central difference approximation rather than use the differentiated Lagrange interpolating polynomials

CHAPTER 4

MODIFIED GENERALIZED FDTD-Q METHOD

Building off the Generalized FDTD-Q method presented in Chapter 2, the technique used to approximate the Laplace operator A is now changed to that of a sixth-order accurate central difference approximation based on the conclusions from Chapter 3. To begin, a sixth-order accurate central difference operator $\frac{1}{\Delta x^2} D_x^2$ is defined, which leads to a seven point central difference approximation of the form

$$\begin{aligned} A\psi^n_{\text{real}}(k) &\approx \frac{1}{\Delta x^2} D_x^2 \psi^n_{\text{real}}(k) \\ &= \frac{1}{180\Delta x^2}[2\psi^n_{\text{real}}(k+3) - 27\psi^n_{\text{real}}(k+2) + 270\psi^n_{\text{real}}(k+1) \\ &\quad - 490\psi^n_{\text{real}}(k) + 270\psi^n_{\text{real}}(k-1) - 27\psi^n_{\text{real}}(k-2) \\ &\quad + 2\psi^n_{\text{real}}(k-3)], \quad \text{with } O(\Delta x^6), \end{aligned} \tag{4.1a}$$

and

$$\begin{aligned} A\psi^n_{\text{imag}}(k) &\approx \frac{1}{\Delta x^2} D_x^2 \psi^n_{\text{imag}}(k) \\ &= \frac{1}{180\Delta x^2}[2\psi^n_{\text{imag}}(k+3) - 27\psi^n_{\text{imag}}(k+2) + 270\psi^n_{\text{imag}}(k+1) \\ &\quad - 490\psi^n_{\text{imag}}(k) + 270\psi^n_{\text{imag}}(k-1) - 27\psi^n_{\text{imag}}(k-2) \\ &\quad + 2\psi^n_{\text{imag}}(k-3)], \quad \text{with } O(\Delta x^6) \end{aligned} \tag{4.1b}$$

A graphical representation of this stencil was shown previously in Figure 3.3.

The Generalized FDTD-Q method restated using the sixth-order central difference approximation of the Laplace operator A is

$$\psi^{n}_{\text{real}}(k) = \psi^{n-1}_{\text{real}}(k) + 2\sum_{p=0}^{N}(\frac{\Delta t}{2})^{2p+1}\frac{(-1)^{p+1}}{(2p+1)!}(\frac{\hbar}{2m}\frac{1}{\Delta x^2}D_x^2 - \frac{V}{\hbar})^{2p+1}\psi^{n-\frac{1}{2}}_{\text{imag}}(k), \quad (4.2a)$$

$$\psi^{n+\frac{1}{2}}_{\text{imag}}(k) = \psi^{n-\frac{1}{2}}_{\text{imag}}(k) + 2\sum_{p=0}^{N}(\frac{\Delta t}{2})^{2p+1}\frac{(-1)^{p}}{(2p+1)!}(\frac{\hbar}{2m}\frac{1}{\Delta x^2}D_x^2 - \frac{V}{\hbar})^{2p+1}\psi^{n}_{\text{real}}(k). \quad (4.2b)$$

Using the sixth-order accurate central difference operator D_x^2, it must now be shown that this operator produces more accurate method, and either an unconditionally stable or conditionally stable method. If the resulting method is conditionally stable under what condition the scheme remains stable must be shown. The order of accuracy of the scheme when using the sixth-order accurate central difference approximation of the Laplace operator must also be shown, and in conclusion the computational algorithm associated with this method will be presented and analyzed.

4.1 Order of Accuracy

To derive the order of accuracy of the Generalized FDTD-Q scheme when using sixth-order accurate central differences, on must begin with the foundations of the Generalized FDTD-Q method, which are Taylor series expansions about various points in time. From Dai and Moxley [11], Equation (2.2) from the original FDTD-Q method is rewritten as

$$\frac{\partial\psi_{\text{real}}(x,t)}{\partial t} = (-\frac{\hbar}{2m}A + \frac{V}{\hbar})\psi_{\text{imag}}(x,t), \quad (4.3a)$$

$$\frac{\partial\psi_{\text{imag}}(x,t)}{\partial t} = (\frac{\hbar}{2m}A - \frac{V}{\hbar})\psi_{\text{real}}(x,t), \quad (4.3b)$$

where $A = \frac{\partial^2}{\partial x^2}$, and Taylor series are used to expand $\psi_{\text{real}}(x,t_n)$ and $\psi_{\text{real}}(x,t_{n-1})$ about $t = t_{n-\frac{1}{2}} = (n-\frac{1}{2})\Delta t$. To avoid clutter in the following derivation and enhance clarity, a few simplifications to Equations (4.3a) and (4.3a) are made. Because the Taylor expansions are in time, let $f(t) = \psi_{\text{real}}(x,t)$ and $g(t) = \psi_{\text{imag}}(x,t)$. Also, let

$W = (\frac{\hbar}{2m}A - \frac{V}{\hbar})$, allowing Equations (4.3a) and (4.3b) to be restated as

$$\frac{\partial f(t)}{\partial t} = -W \cdot g(t). \tag{4.4a}$$

$$\frac{\partial g(t)}{\partial t} = W \cdot f(t). \tag{4.4b}$$

Expanding $f(t_n)$ using a Taylor series about $t = t_{n-\frac{1}{2}}$

$$f(t_n) = f(t_{n-\frac{1}{2}}) + f'(t_{n-\frac{1}{2}})(t_n - t_{n-\frac{1}{2}}) + \frac{f''(t_{n-\frac{1}{2}})}{2!}(t_n - t_{n-\frac{1}{2}})^2 + \cdots + \frac{f^{(M)}(t_{n-\frac{1}{2}})}{M!}(t_n - t_{n-\frac{1}{2}})^M,$$

and recognizing that $t_n - t_{n-\frac{1}{2}} = n\Delta t - n\Delta t + \frac{\Delta t}{2} = \frac{\Delta t}{2}$. Then $f(t_n)$ may be simplified to

$$f(t_n) = f(t_{n-\frac{1}{2}}) + f'(t_{n-\frac{1}{2}})\left(\frac{\Delta t}{2}\right) + \frac{f''(t_{n-\frac{1}{2}})}{2!}\left(\frac{\Delta t}{2}\right)^2 + \cdots + \frac{f^{(M)}(t_{n-\frac{1}{2}})}{M!}\left(\frac{\Delta t}{2}\right)^M. \tag{4.5}$$

Expanding $f(t_{n-1})$ about $t = t_{n-\frac{1}{2}}$ results in

$$f(t_{n-1}) = f(t_{n-\frac{1}{2}}) + f'(t_{n-\frac{1}{2}})(t_{n-1} - t_{n-\frac{1}{2}}) + \frac{f''(t_{n-\frac{1}{2}})}{2!}(t_{n-1} - t_{n-\frac{1}{2}})^2 + \cdots + \frac{f^{(M)}(t_{n-\frac{1}{2}})}{M!}(t_{n-1} - t_{n-\frac{1}{2}})^M,$$

where $t_{n-1} - t_{n-\frac{1}{2}} = n\Delta t - \Delta t - n\Delta t + \frac{\Delta t}{2} = -\frac{\Delta t}{2}$, simplifying to

$$f(t_{n-1}) = f(t_{n-\frac{1}{2}}) - f'(t_{n-\frac{1}{2}})\left(\frac{\Delta t}{2}\right) + \frac{f''(t_{n-\frac{1}{2}})}{2!}\left(\frac{\Delta t}{2}\right)^2 + \cdots + \frac{f^{(M)}(t_{n-\frac{1}{2}})}{M!}\left(-\frac{\Delta t}{2}\right)^M \tag{4.6}$$

Subtracting Equation (4.6) from (4.5)

$$\begin{aligned} f(t_n) - f(t_{n-1}) &= f(t_{n-\frac{1}{2}}) - f(t_{n-\frac{1}{2}}) + f'(t_{n-\frac{1}{2}})\left(\frac{\Delta t}{2}\right) + f'(t_{n-\frac{1}{2}})\left(\frac{\Delta t}{2}\right) \\ &\quad + \frac{f''(t_{n-\frac{1}{2}})}{2!}\left(\frac{\Delta t}{2}\right)^2 - \frac{f''(t_{n-\frac{1}{2}})}{2!}\left(\frac{\Delta t}{2}\right)^2 \\ &\quad + \cdots + \frac{f^M(t_{n-\frac{1}{2}})}{M!}\left(\frac{\Delta t}{2}\right)^M - \frac{f^M(t_{n-\frac{1}{2}})}{M!}\left(-\frac{\Delta t}{2}\right)^M \\ &= 2f'(t_{n-\frac{1}{2}})\left(\frac{\Delta t}{2}\right) + 2\frac{f^{(3)}(t_{n-\frac{1}{2}})}{3!}\left(\frac{\Delta t}{2}\right)^3 \\ &\quad + \cdots + 2\frac{f^{(2M+1)}(t_{n-\frac{1}{2}})}{(2M+1)!}\left(\frac{\Delta t}{2}\right)^{2M+1} \end{aligned} \tag{4.7}$$

$$f(t_n) = f(t_{n-1}) + 2\sum_{p=0}^{\infty}\left(\frac{\Delta t}{2}\right)^{2p+1}\frac{1}{(2p+1)!}\frac{\partial^{(2p+1)} f(t_{n-\frac{1}{2}})}{\partial t^{(2p+1)}} \tag{4.8}$$

Using Equation (4.4) we may now evaluate the derivatives in the above equation for $f(t_n)$ by repeatedly using both Equation (4.4a) and Equation (4.4b):

$$\frac{\partial f(t_{n-\frac{1}{2}})}{\partial t} = -W \cdot g(t_{n-\frac{1}{2}}), \tag{4.9}$$

$$\begin{aligned} \frac{\partial^2 f(t_{n-\frac{1}{2}})}{\partial t^2} &= \frac{\partial}{\partial t}\frac{\partial f(t_{n-\frac{1}{2}})}{\partial t} \\ &= -W\frac{\partial}{\partial t} g(t_{n-\frac{1}{2}}) \\ &= -W^2 f(t_{n-\frac{1}{2}}), \end{aligned} \tag{4.10}$$

$$\begin{aligned} \frac{\partial^3 f(t_{n-\frac{1}{2}})}{\partial t^3} &= \frac{\partial}{\partial t}\frac{\partial^2 f(t_{n-\frac{1}{2}})}{\partial t^2} \\ &= -W^2\frac{\partial}{\partial t} f(t_{n-\frac{1}{2}}) \\ &= W^3 g(t_{n-\frac{1}{2}}), \end{aligned} \tag{4.11}$$

$$\begin{aligned} \frac{\partial^4 f(t_{n-\frac{1}{2}})}{\partial t^4} &= \frac{\partial}{\partial t}\frac{\partial^3 f(t_{n-\frac{1}{2}})}{\partial t^3} \\ &= W^3\frac{\partial}{\partial t} g(t_{n-\frac{1}{2}}) \end{aligned}$$

$$= W^4 f(t_{n\ \frac{1}{2}}), \tag{4 12}$$

$$\frac{\partial^5 f(t_{n-\frac{1}{2}})}{\partial t^5} = \frac{\partial}{\partial t}\frac{\partial^4 f(t_{n\ \frac{1}{2}})}{\partial t^4}$$

$$= W^4 \frac{\partial}{\partial t} f(t_{n-\frac{1}{2}})$$

$$= -W^5 g(t_{n-\frac{1}{2}}) \tag{4 13}$$

up to any order desired Note in the above derivations, the even derivatives are shown solely to enhance clarity Undoing the substitutions made in Equation (4 4), and substituting the above derivatives back into Equation (4 8) yields

$$f(t_n) = f(t_{n-1}) + 2\sum_{p=0}^{N}\left(\frac{\Delta t}{2}\right)^{2p+1}\frac{(-1)^{p+1}}{(2p+1)!}W^{2p+1}g(t_{n-\frac{1}{2}}) + O(\Delta t^{2N+3})$$

$$\psi^n_{\text{real}}(k) = \psi^{n-1}_{\text{real}}(k) + 2\sum_{p=0}^{N}\left(\frac{\Delta t}{2}\right)^{2p+1}\frac{(-1)^{p+1}}{(2p+1)!}\left(\frac{\hbar}{2m}A - \frac{V}{\hbar}\right)^{2p+1}\psi^{n-\frac{1}{2}}_{\text{imag}}(k)$$

$$+ O(\Delta t^{2N+3}) \tag{4 14}$$

The above equation is the Generalized FDTD-Q method for the real component of the wavefunction shown in Equation (2 9a) The key to determining the order of accuracy is to return to the evaluation of the derivatives in Equations (4 9)-(4 13) and the original Taylor series expansion in Equation (4 7)

With the Laplace operator approximated by the sixth-order accurate central difference approximation D_x^2, then the simplification Equations (4 4), may be written including truncation error as

$$\frac{\partial f(t)}{\partial t} = -W g(t) + O(\Delta x^6) \tag{4 15a}$$

$$\frac{\partial g(t)}{\partial t} = W f(t) + O(\Delta x^6) \tag{4 15b}$$

A caveat when using the Laplace operator repeatedly e g to obtain a fourth-order derivative the Laplace operator is recursively applied twice is that

as the derivative order increases, the associated order of accuracy of the spatial derivative does not increase There are several reasons one does not wish to employ increasingly high-order accurate central differences, the primary one being that the number of uncomputable points increases rapidly, and the second being the increased complexity of the requisite stability analysis and computational complexity The result is that when the Laplace operator D_x^2 is raised to some power the order of accuracy remains the same, i e , $O(\Delta x^6)$, which implies that the derivative evaluations, Equations (4 9) (4 13), may be rewritten as

$$\frac{\partial f(t_{n-\frac{1}{2}})}{\partial t} = -Wg(t_{n\ \frac{1}{2}}) + O(\Delta x^6), \tag{4 16}$$

$$\frac{\partial^3 f(t_{n-\frac{1}{2}})}{\partial t^3} = W^3 g(t_{n\ \frac{1}{2}}) + O(\Delta x^6), \tag{4 17}$$

$$\frac{\partial^5 f(t_{n-\frac{1}{2}})}{\partial t^5} = -W^5 g(t_{n-\frac{1}{2}}) + O(\Delta x^6) \tag{4 18}$$

Substituting these equations back into Equation (4 7)

$$f(t_n) = f(t_{n\ 1}) + 2\sum_{p=0}^{N}\left[\left(\frac{\Delta t}{2}\right)^{2p+1}\frac{(-1)^{p+1}}{(2p+1)!}W^{2p+1}g(t_{n\ \frac{1}{2}}) + O(\Delta x^6\Delta t^{2p+1})\right]$$

$$+ O(\Delta t^{2N+3}),$$

$$\psi^n_{\text{real}}(k) = \psi^{n\ 1}_{\text{real}}(k) + 2\sum_{p-0}^{N}\left(\frac{\Delta t}{2}\right)^{2p+1}\frac{(-1)^{p+1}}{(2p+1)!}\left(\frac{\hbar}{2m}\frac{1}{\Delta x^2}D_x^2 - \frac{V}{\hbar}\right)^{2p+1}\psi^{n\ \frac{1}{2}}_{\text{imag}}(k)$$

$$+\sum_{p=0}^{N} O(\Delta x^6\Delta t^{2p+1}) + O(\Delta t^{2N+3}) \tag{4 19}$$

Similarly, employing the Taylor series method to expand $\psi_{\text{imag}}(t_{n+\frac{1}{2}})$ and $\psi_{\text{imag}}(t_{n\ \frac{1}{2}})$ about $t = t_n$, or in terms of the simplified equations expand $g(t_{n+\frac{1}{2}})$ and $g(t_{n+\frac{1}{2}})$ using a Taylor series about $t = t_n$ Beginning with $g(t_{n+\frac{1}{2}})$

$$g(t_{n+\frac{1}{2}}) = g(t_n) + g'(t_n)(t_{n+\frac{1}{2}} - t_n) + \frac{g''(t_n)}{2!}(t_{n+\frac{1}{2}} - t_n)^2$$

$$+ \quad + \frac{f^{(M)}(t_n)}{M!}(t_{n+\frac{1}{2}} - t_n)^M$$

where $t_{n+\frac{1}{2}} - t_n = n\Delta t + \frac{\Delta t}{2} - n\Delta t = \frac{\Delta t}{2}$,

$$g(t_{n+\frac{1}{2}}) = g(t_n) + g'(t_n)\left(\frac{\Delta t}{2}\right) + \frac{g''(t_n)}{2!}\left(\frac{\Delta t}{2}\right)^2 + \cdots + \frac{f^{(M)}(t_n)}{M!}\left(\frac{\Delta t}{2}\right)^M, \tag{4.20}$$

and

$$g(t_{n-\frac{1}{2}}) = g(t_n) + g'(t_n)(t_{n-\frac{1}{2}} - t_n) + \frac{g''(t_n)}{2!}(t_{n-\frac{1}{2}} - t_n)^2 + \cdots + \frac{f^{(M)}(t_n)}{M!}(t_{n-\frac{1}{2}} - t_n)^M,$$

where $t_{n-\frac{1}{2}} - t_n = n\Delta t - \frac{\Delta t}{2} - n\Delta t = -\frac{\Delta t}{2}$,

$$g(t_{n-\frac{1}{2}}) = g(t_n) - g'(t_n)\left(\frac{\Delta t}{2}\right) + \frac{g''(t_n)}{2!}\left(\frac{\Delta t}{2}\right)^2 + \cdots + \frac{f^{(M)}(t_n)}{M!}\left(-\frac{\Delta t}{2}\right)^M. \tag{4.21}$$

Subtracting Equation (4.21) from Equation (4.20), leads to

$$\begin{aligned} g(t_{n+\frac{1}{2}}) - g(t_{n-\frac{1}{2}}) &= g(t_n) - g(t_n) + g'(t_n)\left(\frac{\Delta t}{2}\right) + g'(t_n)\left(\frac{\Delta t}{2}\right) \\ &\quad + \frac{g''(t_n)}{2!}\left(\frac{\Delta t}{2}\right)^2 - \frac{g''(t_n)}{2!}\left(\frac{\Delta t}{2}\right)^2 \\ &\quad + \cdots + \frac{f^{(M)}(t_n)}{M!}\left(\frac{\Delta t}{2}\right)^M - \frac{f^{(M)}(t_n)}{M!}\left(-\frac{\Delta t}{2}\right)^M \\ &= 2g'(t_n)\left(\frac{\Delta t}{2}\right) + 2g^{(3)}(t_n)\left(\frac{\Delta t}{2}\right)^3 \\ &\quad + \cdots + 2\frac{g^{(2M+1)}(t_n)}{(2M+1)!}\left(\frac{\Delta t}{2}\right)^{(2M+1)} \end{aligned}$$

$$g(t_{n+\frac{1}{2}}) = g(t_{n-\frac{1}{2}}) + 2\sum_{p=0}^{\infty}\left(\frac{\Delta t}{2}\right)^{2p+1}\frac{1}{(2p+1)!}\frac{\partial^{(2p+1)}g(t_n)}{\partial t^{(2p+1)}}. \tag{4.22}$$

Again repeatedly using Equation (4.4a) and Equation (4.4b) to evaluate the derivatives in the above equation:

$$\frac{\partial g(t_n)}{\partial t} = W f(t_n). \tag{4.23}$$

$$\begin{aligned}\frac{\partial^2 g(t_n)}{\partial t^2} &= \frac{\partial}{\partial t}\frac{\partial g(t_n)}{\partial t} \\ &= W\frac{\partial}{\partial t} f(t_n) \\ &= -W^2 g(t_n),\end{aligned} \tag{4.24}$$

$$\begin{aligned}\frac{\partial^3 g(t_n)}{\partial t^3} &= \frac{\partial}{\partial t}\frac{\partial^2 g(t_n)}{\partial t^2} \\ &= -W^2\frac{\partial}{\partial t} g(t_n) \\ &= -W^3 f(t_n).\end{aligned} \tag{4.25}$$

$$\begin{aligned}\frac{\partial^4 g(t_n)}{\partial t^4} &= \frac{\partial}{\partial t}\frac{\partial^3 g(t_n)}{\partial t^3} \\ &= -W^3\frac{\partial}{\partial t} f(t_n) \\ &= W^4 g(t_n).\end{aligned} \tag{4.26}$$

$$\begin{aligned}\frac{\partial^5 g(t_n)}{\partial t^5} &= \frac{\partial}{\partial t}\frac{\partial^4 g(t_n)}{\partial t^4} \\ &= W^4\frac{\partial}{\partial t} g(t_n) \\ &= W^5 f(t_n)\end{aligned} \tag{4.27}$$

Substituting the above derivatives back into Equation (4.22) gives

$$g(t_{n+\frac{1}{2}}) = g(t_{n-\frac{1}{2}}) + 2\sum_{p=0}^{N}(\frac{\Delta t}{2})^{2p+1}\frac{(-1)^p}{(2p+1)!}W^{2p+1} f(t_n) + O(\Delta t^{2N+3}).$$

$$\psi_{\text{imag}}(x.\, t_{n+\frac{1}{2}}) = \psi_{\text{imag}}(x.\, t_{n-\frac{1}{2}}) + 2\sum_{p=0}^{N}(\frac{\Delta t}{2})^{2p+1}\frac{(-1)^p}{(2p+1)!}(\frac{\hbar}{2m}A - \frac{V}{\hbar})^{2p+1}\psi_{\text{real}}(x, t_n)$$

$$+ O(\Delta t^{2N+3}) \tag{4.28}$$

Which again is the form of the Generalized FDTD Q method for the imaginary component

Rewriting the derivative expansions to include the truncation error term for the spatial derivative, and recognizing that the order of accuracy will not increase with the order of the derivative, will lead to

$$\frac{\partial g(t_n)}{\partial t} = Wf(t_n) + O(\Delta x^6), \tag{4 29}$$

$$\frac{\partial^3 g(t_n)}{\partial t^3} = -W^3 f(t_n) + O(\Delta x^6), \tag{4 30}$$

$$\frac{\partial^5 g(t_n)}{\partial t^5} = W^5 f(t_n) + O(\Delta x^6) \tag{4 31}$$

Substituting Equations (4 29)–(4 31) back into Equation (4 22) yields the finite difference including truncation error

$$g(t_{n+\frac{1}{2}}) = g(t_{n-\frac{1}{2}}) + 2\sum_{p=0}^{N}\left[(\frac{\Delta t}{2})^{2p+1}\frac{(-1)^p}{(2p+1)!}W^{2p+1}f(t_n) + O(\Delta x^6 \Delta t^{2p+1})\right]$$

$$+ O(\Delta t^{2N+3}),$$

$$\psi_{\text{imag}}^{n+\frac{1}{2}}(k) = \psi_{\text{imag}}^{n-\frac{1}{2}}(k) + 2\sum_{p=0}^{N}(\frac{\Delta t}{2})^{2p+1}\frac{(-1)^p}{(2p+1)!}(\frac{\hbar}{2m}\frac{1}{\Delta x^2}D_x^2 - \frac{V}{\hbar})^{2p+1}\psi_{\text{real}}^{n}(k)$$

$$+ \sum_{p=0}^{N} O(\Delta x^6 \Delta t^{2p+1}) + O(\Delta t^{2N+3}) \tag{4 32}$$

In conclusion, one may algebraically manipulate Equation (4 19) and Equation (4 32) into the following form, and arrive at the Generalized FDTD Q method when used with a sixth-order accurate central difference approximation of the Laplace operator

$$\frac{\psi_{\text{real}}^{n}(k) - \psi_{\text{real}}^{n-1}(k)}{\Delta t} = \sum_{p=0}^{N}\left(\frac{\Delta t}{2}\right)^{2p}\frac{(-1)^{p+1}}{(2p+1)!}\left(\frac{\hbar}{2m}\frac{1}{\Delta x^2}D_x^2 - \frac{V}{\hbar}\right)^{2p+1}\psi_{\text{imag}}^{n-\frac{1}{2}}(k)$$

$$+ \sum_{p=0}^{N} O(\Delta x^6 \Delta t^{2p}) + O(\Delta t^{2N+2}) \tag{4 33a}$$

and

$$\frac{\psi_{\text{imag}}^{n+\frac{1}{2}}(k) - \psi_{\text{imag}}^{n-\frac{1}{2}}(k)}{\Delta t} = \sum_{p=0}^{N} \left(\frac{\Delta t}{2}\right)^{2p} \frac{(-1)^p}{(2p+1)!} \left(\frac{\hbar}{2m}\frac{1}{\Delta x^2} D_x^2 - \frac{V}{\hbar}\right)^{2p+1} \psi_{\text{real}}^{n}(k)$$

$$+ \sum_{p=0}^{N} O(\Delta x^6 \Delta t^{2p}) + O(\Delta t^{2N+2}) \tag{4.33b}$$

The order of accuracy may now be explicitly stated as

$$O(\Delta x^6 + \Delta x^6 \Delta t^2 + \Delta x^6 \Delta t^4 + \cdots + \Delta x^6 \Delta t^{2N} + \Delta t^{2N+2}). \tag{4.34}$$

4.2 Stability

Having shown the order of accuracy of the Generalized FDTD-Q method when the Laplace operator is approximated by sixth-order accurate central differences, the focus now is on whether the sixth-order accurate central difference approximation of the Laplace operator produces a stable FDTD method. In this context stability means that as time progresses the error in the numerical scheme does not grow unbounded. To begin, assume that V is a constant for simplicity, and the Von Neumann analysis [22] is used to analyze the stability of the Generalized FDTD scheme. Let $\psi_{\text{real}}^{n}(k) = \lambda_{\text{real}}^{n} e^{\imath k\beta\Delta x}$ and $\psi_{\text{imag}}^{n+\frac{1}{2}}(k) = \lambda_{\text{imag}}^{n} e^{\imath k\beta\Delta x}$, where λ_{real} and λ_{imag} are amplification factors for $\psi_{\text{real}}^{n}(k)$ and $\psi_{\text{imag}}^{n+\frac{1}{2}}(k)$ respectively. Using the Von Neumann analysis, if one can show that the amplification factors remain bounded, i.e., $\lambda \leq 1$, then this implies that error in the system does not grow over time, and hence the method is stable. Substituting these relations into Equation (4.1) yields

$$A\psi_{\text{real}}^{n}(k) = \frac{1}{180\Delta x^2}[2\lambda_{\text{real}}^{n} e^{\imath(k+3)\beta\Delta x} - 27\lambda_{\text{real}}^{n} e^{\imath(k+2)\beta\Delta x} + 270\lambda_{\text{real}}^{n} e^{\imath(k+1)\beta\Delta x}$$

$$+ 2\lambda_{\text{real}}^{n} e^{\imath(k-3)\beta\Delta x} - 27\lambda_{\text{real}}^{n} e^{\imath(k-2)\beta\Delta x} + 270\lambda_{\text{real}}^{n} e^{\imath(k-1)\beta\Delta x}$$

$$- 490\lambda_{\text{real}}^{n} e^{\imath k\beta\Delta x}]$$

$$= \frac{1}{180\Delta x^2}[2e^{3\imath\beta\Delta x} - 27e^{2\imath\beta\Delta x} + 270e^{\imath\beta\Delta x} - 490$$
$$+ 2e^{-3\imath\beta\Delta x} - 27e^{-2\imath\beta\Delta x} + 270e^{-\imath\beta\Delta x}]\lambda^n_{\text{real}}e^{\imath k\beta\Delta x}.$$

Recalling the relation from Euler's identity

$$\cos(\theta) = \frac{1}{2}\left(e^{\imath\theta} + e^{-\imath\theta}\right), \tag{4.35}$$

and the trigonometric identities

$$\cos(2\theta) = \cos^2(\theta) - \sin^2(\theta), \text{ and} \tag{4.36}$$

$$1 = \cos^2(\theta) + \sin^2(\theta). \tag{4.37}$$

One is able to express the Laplace operator A as

$$A\psi^n_{\text{real}}(k) = \frac{1}{180\Delta x^2}[4\cos(3\beta\Delta x) - 54\cos(2\beta\Delta x)$$
$$+ 540\cos(\beta\Delta x) - 490]\lambda^n_{\text{real}}e^{\imath k\beta\Delta x}$$
$$= \frac{1}{180\Delta x^2}\left[4[\cos^2(3\beta\Delta x/2) - \sin^2(3\beta\Delta x/2)]\right.$$
$$-54[\cos^2(\beta\Delta x) - \sin^2(\beta\Delta x)]$$
$$\left.+540[\cos^2(\beta\Delta x/2) - \sin^2(\beta\Delta x/2)] - 490\right]\lambda^n_{\text{real}}e^{\imath k\beta\Delta x}$$
$$= \frac{1}{180\Delta x^2}\left[4[1 - 2\sin^2(3\beta\Delta x/2)] - 54[1 - 2\sin^2(\beta\Delta x)]\right.$$
$$\left.+540[1 - 2\sin^2(\beta\Delta x/2)] - 490\right]\lambda^n_{\text{real}}e^{\imath k\beta\Delta x}$$
$$= \frac{1}{180\Delta x^2}\left[-8\sin^2(3\beta\Delta x/2) + 108\sin^2(\beta\Delta x)\right.$$
$$\left.-1080\sin^2(\beta\Delta x/2)\right]\lambda^n_{\text{real}}e^{\imath k\beta\Delta x}. \tag{4.38}$$

Reducing Equation (4.38) such that all waves are a combination of $\sin(\beta\Delta x/2)$ is quite tedious. To begin, recognize that

$$\sin(\beta\Delta x) = 2\sin(\beta\Delta x/2)\cos(\beta\Delta x/2) \tag{4.39}$$

and

$$\begin{aligned}\sin^2(\beta\Delta x) &= 4\sin^2(\beta\Delta x/2)\cos^2(\beta\Delta x/2)\\ &= 4\sin^2(\beta\Delta x/2)[1-\sin^2(\beta\Delta x/2)]\\ &= 4\sin^2(\beta\Delta x/2) - 4\sin^4(\beta\Delta x/2).\end{aligned} \tag{4.40}$$

Reducing $\sin^2(3\beta\Delta x/2)$ is done in a similar, yet more complicated fashion

$$\begin{aligned}\sin^2(3\beta\Delta x/2) &= [\sin(\beta\Delta x/2+\beta\Delta x)]^2\\ &= [\sin(\beta\Delta x/2)\cos(\beta\Delta x)+\sin(\beta\Delta x)\sin(\beta\Delta x/2)]^2\\ &= \sin^2(\beta\Delta x/2)\cos^2(\beta\Delta x)+\sin^2(\beta\Delta x)\cos^2(\beta\Delta x/2)\\ &\quad + 2\sin(\beta\Delta x/2)\cos(\beta\Delta x)\sin(\beta\Delta x)\cos(\beta\Delta x/2)\end{aligned} \tag{4.41}$$

From Equation (4.41) recognizing that

$$\begin{aligned}\cos(\beta\Delta x) &= \cos^2(\beta\Delta x/2)-\sin^2(\beta\Delta x/2)\\ &= 1-2\sin^2(\beta\Delta x/2)\end{aligned} \tag{4.42}$$

and

$$\cos^2(\beta\Delta x) = \left[1-2\sin^2(\beta\Delta x/2)\right]^2, \tag{4.43}$$

then Equation (4.41) may be rewritten as

$$\begin{aligned}\sin^2(3\beta\Delta x/2) &= \sin^2(\beta\Delta x/2)\left[1-2\sin^2(\beta\Delta x/2)\right]^2\\ &\quad + \left[4\sin^2(\beta\Delta x/2)-4\sin^4(\beta\Delta x/2)\right]\left[1-\sin^2(\beta\Delta x/2)\right]\end{aligned}$$

$$
\begin{aligned}
&+ 4\sin^2(\beta\Delta x/2)\cos^2(\beta\Delta x/2)\left[1 - 2\sin^2(\beta\Delta x/2)\right] \\
&= \sin^2(\beta\Delta x/2)\left[1 - 4\sin^2(\beta\Delta x/2) + 4\sin^4(\beta\Delta x/2)\right] \\
&\quad + 4\sin^2(\beta\Delta x/2) - 8\sin^4(\beta\Delta x/2) + 4\sin^6(\beta\Delta x/2) \\
&\quad + 4\sin^2(\beta\Delta x/2)\cos^2(\beta\Delta x/2)\left[1 - 2\sin^2(\beta\Delta x/2)\right] \\
&= \sin^2(\beta\Delta x/2) - 4\sin^4(\beta\Delta x/2) + 4\sin^6(\beta\Delta x/2) \\
&\quad + 4\sin^2(\beta\Delta x/2) - 8\sin^4(\beta\Delta x/2) + 4\sin^6(\beta\Delta x/2) \\
&\quad + \cos^2(\beta\Delta x/2)\left[4\sin^2(\beta\Delta x/2) - 8\sin^4(\beta\Delta x/2)\right] \\
&= 5\sin^2(\beta\Delta x/2) - 12\sin^4(\beta\Delta x/2) + 8\sin^6(\beta\Delta x/2) \\
&\quad + \left[1 - \sin^2(\beta\Delta x/2)\right]\left[4\sin^2(\beta\Delta x/2) - 8\sin^4(\beta\Delta x/2)\right] \\
&= 5\sin^2(\beta\Delta x/2) - 12\sin^4(\beta\Delta x/2) + 8\sin^6(\beta\Delta x/2) \\
&\quad + 4\sin^2(\beta\Delta x/2) - 12\sin^4(\beta\Delta x/2) + 8\sin^6(\beta\Delta x/2) \\
&= 9\sin^2(\beta\Delta x/2) - 24\sin^4(\beta\Delta x/2) + 16\sin^6(\beta\Delta x/2). \qquad (4.44)
\end{aligned}
$$

Collecting the terms from Equation (4.40) and Equation (4.44), and substituting back into Equation (4.38), one obtains

$$
\begin{aligned}
A\psi^n_{\text{real}}(k) &= -\frac{4}{180\Delta x^2}\left[180\sin^2(\beta\Delta x/2) + 60\sin^4(\beta\Delta x/2)\right. \\
&\qquad \left. +32\sin^6(\beta\Delta x/2)\right]\lambda^n_{\text{real}} e^{\imath k\beta\Delta x} \\
&= -\frac{4}{45\Delta x^2}\left[45\sin^2(\beta\Delta x/2) + 15\sin^4(\beta\Delta x/2)\right. \\
&\qquad \left. +8\sin^6(\beta\Delta x/2)\right]\lambda^n_{\text{real}} e^{\imath k\beta\Delta x} \qquad (4.45)
\end{aligned}
$$

A similar analysis of $A\psi^n_{\text{imag}}(k)$ leads to the following equations expressed in

terms of the sixth-order accurate central difference operator D_x^2

$$\frac{1}{\Delta t^2} D_x^2 \psi_{\text{real}}^n(k) = \frac{1}{\Delta x^2}\left[-\frac{4}{45}\left(45\sin^2\frac{\beta\Delta x}{2} + 15\sin^4\frac{\beta\Delta x}{2} + 8\sin^6\frac{\beta\Delta x}{2}\right)\right]\lambda_{\text{real}}^n e^{\imath k\beta\Delta x}. \tag{4.46a}$$

$$\frac{1}{\Delta x^2} D_x^2 \psi_{\text{imag}}^{n+\frac{1}{2}}(k) = \frac{1}{\Delta x^2}\left[-\frac{4}{45}\left(45\sin^2\frac{\beta\Delta x}{2} + 15\sin^4\frac{\beta\Delta x}{2} + 8\sin^6\frac{\beta\Delta x}{2}\right)\right]\lambda_{\text{imag}}^n e^{\imath k\beta\Delta x}. \tag{4.46b}$$

To simplify notation in the following equations, let

$$Q = \frac{4}{45}\left[45\sin^2(\beta\Delta x/2) + 15\sin^4(\beta\Delta x/2) + 8\sin^6(\beta\Delta x/2)\right]. \tag{4.47}$$

and Equation (4.46) may then be more compactly stated as

$$\frac{1}{\Delta x^2} D_x^2 \psi_{\text{real}}^n(k) = -\frac{1}{\Delta x^2} Q\lambda_{\text{real}}^n e^{\imath k\beta\Delta x}, \tag{4.48a}$$

$$\frac{1}{\Delta x^2} D_x^2 \psi_{\text{imag}}^{n+\frac{1}{2}}(k) = -\frac{1}{\Delta x^2} Q\lambda_{\text{imag}}^n e^{\imath k\beta\Delta x}. \tag{4.48b}$$

Returning to the Generalized FDTD-Q method presented at the beginning of the chapter, Equation (4.2) is restated in terms of the error amplifications

$$\begin{aligned}\lambda_{\text{real}}^n &= \lambda_{\text{real}}^{n-1} + 2\sum_{p=0}^{N}\left(\frac{\Delta t}{2}\right)^{2p+1}\frac{(-1)^{p+1}}{(2p+1)!}\left[-\frac{\hbar}{2m}\frac{1}{\Delta x^2}Q - \frac{V}{\hbar}\right]^{2p+1}\lambda_{\text{imag}}^{n-1} \\ &= \lambda_{\text{real}}^{n-1} + 2\sum_{p=0}^{N}\frac{(-1)^{p}}{(2p+1)!}\left[\frac{\hbar}{4m}\frac{\Delta t}{\Delta x^2}Q - \frac{V\Delta t}{2\hbar}\right]^{2p+1}\lambda_{\text{imag}}^{n-1}.\end{aligned} \tag{4.49a}$$

$$\begin{aligned}\lambda_{\text{imag}}^n &= \lambda_{\text{imag}}^{n-1} + 2\sum_{p=0}^{N}\left(\frac{\Delta t}{2}\right)^{2p+1}\frac{(-1)^{p}}{(2p+1)!}\left[-\frac{\hbar}{2m}\frac{1}{\Delta x^2}Q - \frac{V}{\hbar}\right]^{2p+1}\lambda_{\text{real}}^{n} \\ &= \lambda_{\text{imag}}^{n-1} - 2\sum_{p=0}^{N}\frac{(-1)^{p}}{(2p+1)!}\left[\frac{\hbar}{4m}\frac{\Delta t}{\Delta x^2}Q - \frac{V\Delta t}{2\hbar}\right]^{2p+1}\lambda_{\text{real}}^{n}\end{aligned} \tag{4.49b}$$

These equations may be more compactly expressed as

$$\lambda^n_{\text{real}} = \lambda^{n-1}_{\text{real}} + \alpha\lambda^{n-1}_{\text{imag}}, \tag{4.50a}$$

$$\lambda^n_{\text{imag}} = \lambda^{n-1}_{\text{imag}} - \alpha\lambda^{n}_{\text{real}}, \tag{4.50b}$$

where $\alpha = 2\sum_{p=0}^{N} \frac{(-1)^p}{(2p+1)!}\left[\frac{\hbar}{4m}r\,Q - \frac{V\Delta t}{2\hbar}\right]^{2p+1}$ and r is the mesh ratio $\frac{\Delta t}{\Delta x^2}$.

Since Equation (4.50a) is true for any time level n, it may be rewritten as

$$\lambda^{n+1}_{\text{real}} = \lambda^{n}_{\text{real}} + \alpha\lambda^{n}_{\text{imag}} \tag{4.51}$$

Subtracting Equation (4.51) by Equation (4.50a), with the motivation being that as time progresses the difference between the error at different time steps remains constant. The resulting equation

$$\lambda^{n+1}_{\text{real}} - \lambda^{n}_{\text{real}} = \lambda^{n}_{\text{real}} - \lambda^{n-1}_{\text{real}} + \alpha\lambda^{n}_{\text{imag}} - \alpha\lambda^{n-1}_{\text{imag}}, \tag{4.52}$$

may be simplified using Equation (4.50b) leading to a quadratic equation

$$\lambda^{n+1}_{\text{real}} - 2\lambda^{n}_{\text{real}} - \lambda^{n-1}_{\text{real}} = \alpha(\lambda^{n}_{\text{imag}} - \lambda^{n-1}_{\text{imag}}),$$

$$\lambda^{n+1}_{\text{real}} - 2\lambda^{n}_{\text{real}} - \lambda^{n-1}_{\text{real}} = \alpha(\lambda^{n-1}_{\text{imag}} - \alpha\lambda^{n}_{\text{real}} - \lambda^{n-1}_{\text{imag}}),$$

$$\lambda^{n+1}_{\text{real}} - 2\lambda^{n}_{\text{real}} - \lambda^{n-1}_{\text{real}} = -\alpha^2\lambda^{n}_{\text{real}},$$

$$\lambda^{2}_{\text{real}} - (2-\alpha^2)\lambda_{\text{real}} - 1 = 0 \tag{4.53}$$

Recall that λ is an amplification factor, and to have a stable method, these amplification factors must be bounded Using the fact that for a quadratic equation $r^2 + Br + C = 0$. the solution r satisfies $|r| \leq 1$ if and only if $|B| \leq 1 + |C|$ and $|C| \leq 1$. From Equation (4.53) it is clear that $|C| \leq 1$, and to have $|B| \leq 1 + |C|$ then the following relation must be true $|\lambda_{\text{real}}| \leq 1$ if and only if $|\alpha| \leq 2$

By the Von Neumann analysis it is concluded that the Generalized FDTD-Q scheme is stable if $|\alpha| \leq 2$,

$$\left| 2\sum_{p=0}^{N} \frac{(-1)^p}{(2p+1)!} \left[\frac{\hbar}{4m} r \quad Q + \frac{V\Delta t}{2\hbar} \right]^{2p+1} \right| \leq 2,$$

$$\left| \sum_{p=0}^{N} \frac{(-1)^p}{(2p+1)!} \left[\frac{\hbar}{4m} r \quad Q + \frac{V\Delta t}{2\hbar} \right]^{2p+1} \right| \leq 1 \tag{4 54}$$

From Equation (4 54) one can conclude that the Generalized FDTD-Q method is stable, but with the parameters contained in the equation e g , N, m, V, $\hbar$, Δx, and Δt, it is unclear if the scheme is unconditionally stable or if the aforementioned parameters have an impact on the stability First recall the Taylor series representation of a sine wave

Lemma 4.1. *Taylor series representation of a sine wave*

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n-1)!} x^{2n+1} \tag{4 55}$$

Now, suppose $N \to \infty$,

$$\lim_{N\to\infty} \sum_{p=0}^{N} \frac{(-1)^p}{(2p+1)!} \left[\frac{\hbar}{4m} r \quad Q + \frac{V\Delta t}{2\hbar} \right]^{2p+1}$$

$$= \sum_{p=0}^{\infty} \frac{(-1)^p}{(2p+1)!} \left[\frac{\hbar}{4m} r \quad Q + \frac{V\Delta t}{2\hbar} \right]^{2p+1}$$

$$= \sin\left(\frac{\hbar}{4m} r \quad Q + \frac{V\Delta t}{2\hbar} \right) \tag{4 56}$$

It is immediately clear that regardless of the parameters m V, $\hbar$, Δx and Δt, Equation (4 54) is automatically satisfied as $N \to \infty$ implying the scheme is unconditionally stable

However, in practice one may not allow N to be arbitrarily large, and so Equation (4 54) is imposed using the maximum value of $\frac{\hbar}{4m} r$ $Q + \frac{V\Delta t}{2\hbar}$ The maximum

value of Q is

$$\begin{aligned}\max|Q| &= \max\left|\frac{4}{45}\left[45\sin^2(\beta\Delta x/2)+15\sin^4(\beta\Delta x/2)+8\sin^6(\beta\Delta x/2)\right]\right| \\ &= \frac{4}{45}[45+15+8] \\ &= \frac{272}{45}, \end{aligned} \tag{4.57}$$

and the required stability condition is

$$\begin{aligned}\left|\sum_{p=0}^{N}\frac{(-1)^p}{(2p+1)!}\left[\frac{\hbar}{4m}r\cdot\max|Q|+\frac{\Delta t}{2\hbar}\max|V|\right]^{2p+1}\right| &\le c<1 \\ \left|\sum_{p=0}^{N}\frac{(-1)^p}{(2p+1)!}\left[\frac{\hbar}{4m}\frac{272}{45}r+\frac{\Delta t}{2\hbar}\max|V|\right]^{2p+1}\right| &\le c<1 \\ \left|\sum_{p=0}^{N}\frac{(-1)^p}{(2p+1)!}\left[\frac{68\hbar}{45m}r+\frac{\Delta t}{2\hbar}\max|V|\right]^{2p+1}\right| &\le c<1, \end{aligned} \tag{4.58}$$

where c is a constant. Using a similar argument, one may obtain the same inequality as that in Equation (4.58) for λ_{imag}. Hence, one arrives at the following theorem.

Theorem 4.1. *The Generalized FDTD scheme for sixth-order accurate central differences*

$$\psi^{n}_{real}(k)=\psi^{n-1}_{real}(k)+2\sum_{p=0}^{N}\frac{(-1)^{p+1}}{(2p+1)!}\left[\frac{\hbar}{4m}rD_x^2-\frac{V\Delta t}{2\hbar}\right]^{2p+1}\psi^{n-\frac{1}{2}}_{imag}(k). \tag{4.59a}$$

$$\psi^{n+\frac{1}{2}}_{imag}(k)=\psi^{n-\frac{1}{2}}_{imag}(k)+2\sum_{p=0}^{N}\frac{(-1)^{p}}{(2p+1)!}\left[\frac{\hbar}{4m}rD_x^2-\frac{V\Delta t}{2\hbar}\right]^{2p+1}\psi^{n}_{real}(k) \tag{4.59b}$$

is stable if the following the condition is satisfied

$$\left|\sum_{p=0}^{N}\frac{(-1)^p}{(2p+1)!}\left[\frac{\hbar}{4m}\quad\frac{272}{45}r+\frac{\Delta t}{2\hbar}\max|V|\right]^{2p+1}\right|\le 1 \tag{4.60}$$

4.3 Computational Algorithm

Having shown a theoretical basis for utilizing a sixth-order accurate central difference approximation of the Laplace operator, it is essential to bridge the gap between theory and computation. In this section, pseudocode is presented that translates the mathematical methods presented in the previous sections and chapters into a format more suitable for computation, while avoiding the technical implementation details that arise when writing an actual program that solves a real problem The goal behind pseudocode is to present the fundamental concepts behind an algorithm, while not burdening the reader with intricate implementation details that may be specific to solving a unique problem. To this end, the algorithm for the Generalized FDTD-Q method is presented in Algorithm 4.1. Note the algorithm has been summarized in its entirety on a single page, should one wish to view the actual source code as implemented in the FORTRAN77 language, the reader is directed to Appendix C, but it should be noted that the source code presented is specific to solving a model problem presented in Chapter 5.

Algorithm 4.1: Pseudocode for the Generalized Finite-Difference Time-Domain for method.

Input: Grid spacing Δx
Input: Number of grid points N_{points}
Input: Number of time steps N_{steps}
Input: Parameter N
Input: Mesh ratio r
Input: $1 \times N_{\text{points}}$ Array of initial values for the real component of the wavefunction ϕ_{real}
Input: $1 \times N_{\text{points}}$ Array of initial values for the imaginary component of the wavefunction ϕ_{imag}
Input: $1 \times N_{\text{points}}$ Array of initial values for the potential function V_0

// Apply initial conditions
for $k = 0$ **to** N_{points} **do**
    $\psi_{\text{real}}(k) = \phi_{\text{real}}(k)$
    $\psi_{\text{imag}}(k) = \phi_{\text{imag}}(k)$
    $V(k) = V_0(k)$
end
$\Delta t = r \cdot \Delta x^2$
// Begin time stepping loop
for $n = 1$ **to** N_{steps} **do**
    // Compute even-order derivatives of $\psi_{\text{imag}}^{n-\frac{1}{2}}$ up to $4N + 2$
    HighOrderLaplaceDiff($\psi_{\text{imag}}^{n-\frac{1}{2}}, N_{\text{points}}, 4N + 2$)
    // Compute ψ_{real}^{n} using the derivatives of $\psi_{\text{imag}}^{n-\frac{1}{2}}$
    foreach *Computable grid point* k **do**
        //Compute $\psi_{\text{real}}^{n}(k)$ using the Generalized FDTD-Q scheme Eq. (2.9)
    end
    // Compute even-order derivatives of ψ_{real}^{n} up to $4N + 2$
    HighOrderLaplaceDiff($\psi_{\text{real}}^{n}, N_{\text{points}}, 4N + 2$)
    // Compute $\psi_{\text{imag}}^{n+\frac{1}{2}}$ using the derivatives of ψ_{real}^{n}
    foreach *Computable grid point* k **do**
        // Compute $\psi_{\text{imag}}^{n+\frac{1}{2}}(k)$ using the Generalized FDTD-Q scheme Eq. (2.9)
    end
end

From Algorithm 4.1, one can see that the majority of the algorithm is inside of the time loop on Lines 9–22, and it is for this reason algorithms of this type are typically referred to as *timestepping* or *timemarching* algorithms. A hidden detail not shown in Algorithm 4.1 is that both time and space have been discretized into a finite set of grid points. This discretization was introduced in Equations (2.5) and (2.5) in Chapter 2.

The work from Chapter 3 related to numerical differentiation is present on Line 11 and Line 17, where the procedure *HighOrderLaplaceDiff* is used to obtain the high-order derivatives required by the Generalized FDTD-Q scheme. The pseudocode for this procedure is shown in Algorithm 4.2 and related Algorithm 4.3, and one should be aware that the specific approximation of the Laplace operator is left intentionally ambiguous. This ambiguity is essential given one only needs to prove the theoretical basis for using a specific approximation of the Laplace operator, and the Generalized FDTD-Q algorithm will remain valid.

Algorithm 4.2: Pseudocode for obtaining high-order derivatives by recursively applying the Laplace operator.

Function: HighOrderLaplaceDiff($\varphi, N_{\text{points}}, M$)
Input: Number of grid points N_{points}
Input: Highest order derivative desired M
Input: $1 \times N_{\text{points}}$ Array of function values φ
Output: $\frac{M}{2}$ 1-D arrays of size $1 \times N_{\text{points}}$ containing the function φ differentiated up to $\varphi^{(M)}$
begin
  Apply the Laplace operator repeatedly to obtain high-order derivatives
  for $m = 1$ **to** $\frac{M}{2}$ **do**
    $\varphi^{(2m)} = \text{LaplaceDiff}(\varphi^{(2(m-1)}, N_{\text{points}})$
  end
  Return $\varphi^{(2)}, \varphi^{(4)}, \ldots, \varphi^{(M)}$
end

Algorithm 4.3: Pseudocode for applying the Laplace operator.

Function: LaplaceDiff(φ, N_{points})
Input: Number of grid points N_{points}
Input: $1 \times N_{\text{points}}$ Array of function values φ
Output: $1 \times N_{\text{points}}$ Array of the function φ twice differentiated $\varphi^{(2)}$
begin
  Apply the Laplace operator to the values in φ
  foreach *Computable grid point* k **do**
    Compute $\varphi^{(2)}(k)$ using Eq. (3.13).
  end
  Return $\varphi^{(2)}$
end

From the pseudocode presented, enough detail is shown such that one may note the computational complexity of the entire method. From Lines 2–5, one may note that applying the initial conditions will require at least $cN_{\text{points}} + C$ computations. where c and C are constants, and clearly the algorithm will never perform more than $cN_{\text{points}} + C$ computations. Therefore both the upper and lower bounds are asymptotically the same, and it may be stated that the algorithm has computational complexity $\Theta(N_{\text{points}})$. Similar reasoning is implied throughout this analysis, and asymptotic bounds are directly written in Θ notation. The time loop is more complicated, from Line 9 it is clear the loop will be executed $\Theta(N_{\text{steps}})$ times, but the analysis of the interior of the loop is more complicated and rather than approach the loop as a whole, analysis will be performed first on the differentiation procedure from Algorithm 4.2.

The *HighOrderLaplaceDiff* procedure from Algorithm 4.2 requires $\frac{M}{2}$ evaluations of the *LaplaceDiff* procedure. In the context of the Generalized FDTD-Q method. M is actually $2(2N + 1)$ as taken from Line 11 of the Generalized FDTD-Q algorithm in Algorithm 4.1. Using this information one may then realize that the *HighOrderLaplaceDiff* procedure requires $\frac{2(2N+1)}{2} = 2N + 1$ evaluations of the *LaplaceDiff* procedure. To determine the complexity of the *HighOrderLaplaceDiff*

procedure only needs to know the complexity of the *LaplaceDiff* procedure, which may be observed from Algorithm 4.3 noting that the *LaplaceDiff* procedure requires

$$\Theta(N_{\text{points}}) \quad \text{computation.} \tag{4.61}$$

With this information one may then deduce that the *HighOrderLaplaceDiff* procedure requires

$$\Theta((2N+1)N_{\text{points}}) \quad \text{computation.} \tag{4.62}$$

Returning to the complexity of the time loop, one may now model the computational complexity of the time loop as follows, where the interior of the loop has computational complexity

$$\overbrace{2\Theta((2N+1)N_{\text{points}})}^{\text{Line 11 and Line 17}} + \overbrace{2\Theta(N_{\text{points}})}^{\text{Line 13 and Line 19}},$$

$$\Theta(4(N+1)N_{\text{points}}). \tag{4.63}$$

Leading to the computational complexity of the Generalized FDTD-Q algorithm,

$$\overbrace{\Theta(N_{\text{points}})}^{\text{initial conditions}} + \overbrace{\Theta(4(N+1)N_{\text{points}} \cdot N_{\text{steps}})}^{\text{time loop}},$$

$$\Theta(N_{\text{points}} \cdot N_{\text{steps}}). \tag{4.64}$$

Where Equation (4.64) is the result of removing the constants, and retaining only the dominant terms in the equation. This is done because Θ is a model of the asymptotic behavior, i.e., $C_1 N_{\text{points}} + C_2 N_{\text{points}} N_{\text{steps}}$ simplifies to $C_2 N_{\text{points}} N_{\text{steps}}$.

4.4 Summary

In this chapter the theoretical basis for using a sixth-order accurate approximation of the Laplace operator was established: the order of accuracy of the resulting Generalized FDTD-Q scheme was derived; the stability condition imposed by using this sixth-order accurate operator was shown; and the computational algorithm

used to evaluated the scheme was presented and analyzed. The significance of this work may be summarized by stating that the sixth-order accurate Laplace operator has improved the theoretical order of accuracy by four orders of magnitude, while imposing a more stringent stability requirement as shown in Theorem 4.1. The following chapter will evaluate whether these theoretical aspects hold true when used to solve real problems.

CHAPTER 5

RESULTS

Having presented the FDTD-Q and Generalized FDTD-Q methods in Chapter 2; meticulously analyzed various compelling differentiation techniques and selected the most appropriate in Chapter 3, proved in Chapter 4 that the selected differentiation technique is stable when applied to the Generalized FDTD-Q scheme; Now, numerical experiments are performed to demonstrate the superiority of the choice made in Chapter 3 when compared against the FDTD-Q and published Generalized FDTD-Q methods.

To compare against the FDTD-Q and published Generalized FDTD-Q methods two problems have been selected. The first problem has an exact solution, and will be referred to as the *model problem*, because the model problem has an exact solution the absolute error may be computed when using each of the numerical schemes listed above. The model problem also allows one to observe when a scheme becomes divergent, which allows ones to observe the stability through experimentation. Using this model problem, it will be shown that the stability condition presented in Chapter 4 is correct, and that the scheme has absolute error several orders of magnitude smaller than the FDTD-Q and published Generalized FDTD-Q schemes. The second problem chosen has been taken from Sullivan [29] and Dai et al [10], and simulates a particle moving in 1-D free space and then hitting an energy potential.

5.1 The Model Problem

The model problem chosen is a one dimensional time-dependent Schrodinger equation posed as follows

$$\frac{\partial \psi(x,t)}{\partial t} = \imath \frac{\partial^2 \psi(x,t)}{\partial x^2} - \imath V(x,t)\psi(x,t), \quad x \in [a,b], \quad t > 0 \tag{5 1}$$

$$\psi(a,t) = \psi(b\ t) = 0, \quad t > 0,$$

$$V(x,0) = \pi^2,$$

where initial conditions for $\psi(x,t)$ are provided by the complex function $\phi(x)$ The analytical solution of Equation (5 1) is $\psi(x,t) = e^{-\imath 2\pi^2 t}\sin(\pi x)$, and equation is solved over the interval $0 \leq c \leq 1$ and $0 < t \leq 1$ The initial conditions are derived from the analytical solution using Euler's identity $c^{\imath\theta} = \cos(\theta) + \imath\sin(\theta)$

$$\begin{aligned}
\phi(x) &= e^{-\imath 2\pi^2 t}\sin(\pi x) \\
&= \cos(-2\pi^2 t)\sin(\pi x) + \imath\sin(-2\pi^2 t)\sin(\pi x) \\
&= \cos(2\pi^2 t)\sin(\pi x) - \imath\sin(2\pi^2 t)\sin(\pi x)
\end{aligned} \tag{5 2}$$

Note that attempting to compute the initial conditions directly from $\psi(x, t = 0) = e^{-\imath 2\pi^2 t}\sin(\pi x)$ will effectively remove the imaginary component and we seek the nontrivial solution, i e , $\phi_{\text{imag}} \neq 0$ Continuing from Equation (5 2) leads to

$$\phi_{\text{real}}(x) = \cos(2\pi^2 t)\sin(\pi x), \quad \text{and} \quad \varphi_{\text{imag}}(x) = \sin(2\pi^2 t)\sin(\pi x)$$

Allowing $t = 0$ in $\phi_{\text{real}}(x)$ causes no problems, but again one must take care to ensure that the imaginary component does not become zero To handle this recall from the FDTD-Q scheme that ψ_{imag} is computed at time $t + \frac{1}{2}$ using this information recognize that the initial t is $t = \frac{\Delta t}{2}$ leading to the initial conditions

$$\psi(x,0) = \phi(x) = \begin{cases} \phi_{\text{real}}(x) = \sin(\pi x), \\ \phi_{\text{imag}}(x) = -\sin(\pi^2 \Delta t)\sin(\pi x). \end{cases} \tag{5.3}$$

To compare against the original FDTD-Q scheme Equation (2.3), recall that the original FDTD-Q scheme is the Generalized FDTD-Q scheme with $N = 0$. For this analysis there are several cases that the improved Generalized FDTD-Q scheme must be compared against, the parameter N is chosen to be $N = 3$, and the Generalized FDTD-Q scheme with sixth-order accurate spatial derivatives is compared against the following:

- The original second-order accurate FDTD-Q scheme i.e., the published Generalized FDTD-Q scheme with $N = 0$.
- The published Generalized FDTD-Q scheme using second-order accurate spatial derivatives and $N = 3$.
- The original sixth-order accurate FDTD-Q scheme i.e., the Generalized FDTD-Q scheme with sixth-order accurate spatial derivatives and $N = 0$.

5.1.1 Stability Conditions for the Model Problem

The stability condition imposed upon the modified (sixth-order accurate) Generalized FDTD-Q scheme is taken from Theorem 4.1. For the model problem Equation (4.60) takes the form

$$\left| \sum_{p=0}^{N} \frac{(-1)^p}{(2p+1)!} \left[\frac{1}{2} \cdot \frac{272}{45} r + \pi^2 \Delta t \right]^{2p+1} \right| \leq c < 1,$$

$$\left| \sum_{p=0}^{N} \frac{(-1)^p}{(2p+1)!} \left[\frac{136}{45} r + \pi^2 \Delta t \right]^{2p+1} \right| \leq c < 1 \tag{5.4}$$

Recall r is the mesh ratio $r = \frac{\Delta t}{\Delta x^2}$, or $\Delta t = r \cdot \Delta x^2$, replacing Δt by this relation in the above condition allows one to formulate the stability condition entirely in terms of r, N, and Δx

$$\left| \sum_{p=0}^{N} \frac{(-1)^p}{(2p+1)!} \left[\frac{136}{45} r + \pi^2 r \Delta x^2 \right]^{2p+1} \right| \leq c < 1. \tag{5.5}$$

Now choosing a specific N and Δx one may determine the largest possible mesh ratio that will remain stable The grid spacings chosen are $\Delta x = 0\,01$ and $\Delta x = \frac{1}{2}\ 0\,01 = 0\,005$, N is chosen to be $N = 0$ and $N = 3$ First consider the case of $\Delta x = 0\,01$, Equation (5 5) reduces to a polynomial in r

$$\left|0\,4579r^7 - 2\,1045r^5 + 4\,6052r^3 - 3\,0232r\right| < 1 \tag{5 6}$$

Which has a positive real root at

$$r = 1\,254513102904631 \tag{5 7}$$

which leads to the following theorem

Theorem 5.1. *The mesh ratio for the sixth-order accurate Generalized FDTD-Q method with* $\Delta x = 0\,01$ *and* $N = 3$ *will produce a stable method if*

$$|r| < 1\,254513102904631 \tag{5 8}$$

A similar analysis may be performed for $N = 3$ and $\Delta x = 0\,005$, as well as $N = 0$ and $\Delta x = 0\,01$ and $\Delta x = 0\,005$ We call these values the *critical mesh ratios*, as the scheme is only stable if $|r| < \hat{c}$ Notice these critical mesh ratios are strict, and equally implies the stability condition is no longer satisfied The stability condition for the second-order accurate Generalized FDTD-Q scheme from [11] is

$$\left|\sum_{p=0}^{N} \frac{(-1)^p}{(2p+1)!}\left[2r + \pi^2 r\,\Delta x^2\right]^{2p+1}\right| \leq c < 1 \tag{5 9}$$

Following the procedure used to arrive at Theorem 5 1, the critical mesh ratios have been computed for both the second-order accurate Generalized FDTD-Q method and the sixth-order accurate Generalized FDTD-Q method, and are shown in Table 5 1

Table 5 1 Critical mesh ratios $\hat{c}$ for the Generalized FDTD-Q schemes when solving the model problem using $\Delta x = 0\ 01$ or $\Delta x = 0\ 005$, and $N \in \{0, 1, 2, 3\}$

	Critical Mesh Ratios $\hat{c}$ $\lvert r\rvert < \hat{c} \implies$ Stability			
	$O(\Delta x^2)$ Scheme		$O(\Delta x^6)$ Scheme	
N	0 01	0 005	0 01	0 005
0	0 49975	0 49999	0 33077	0 33085
1	1 42295	1 42348	0 94182	0 94205
2	1 83958	1 84026	1 21757	1 21787
3	1 89539	1 89609	1 25451	1 25482

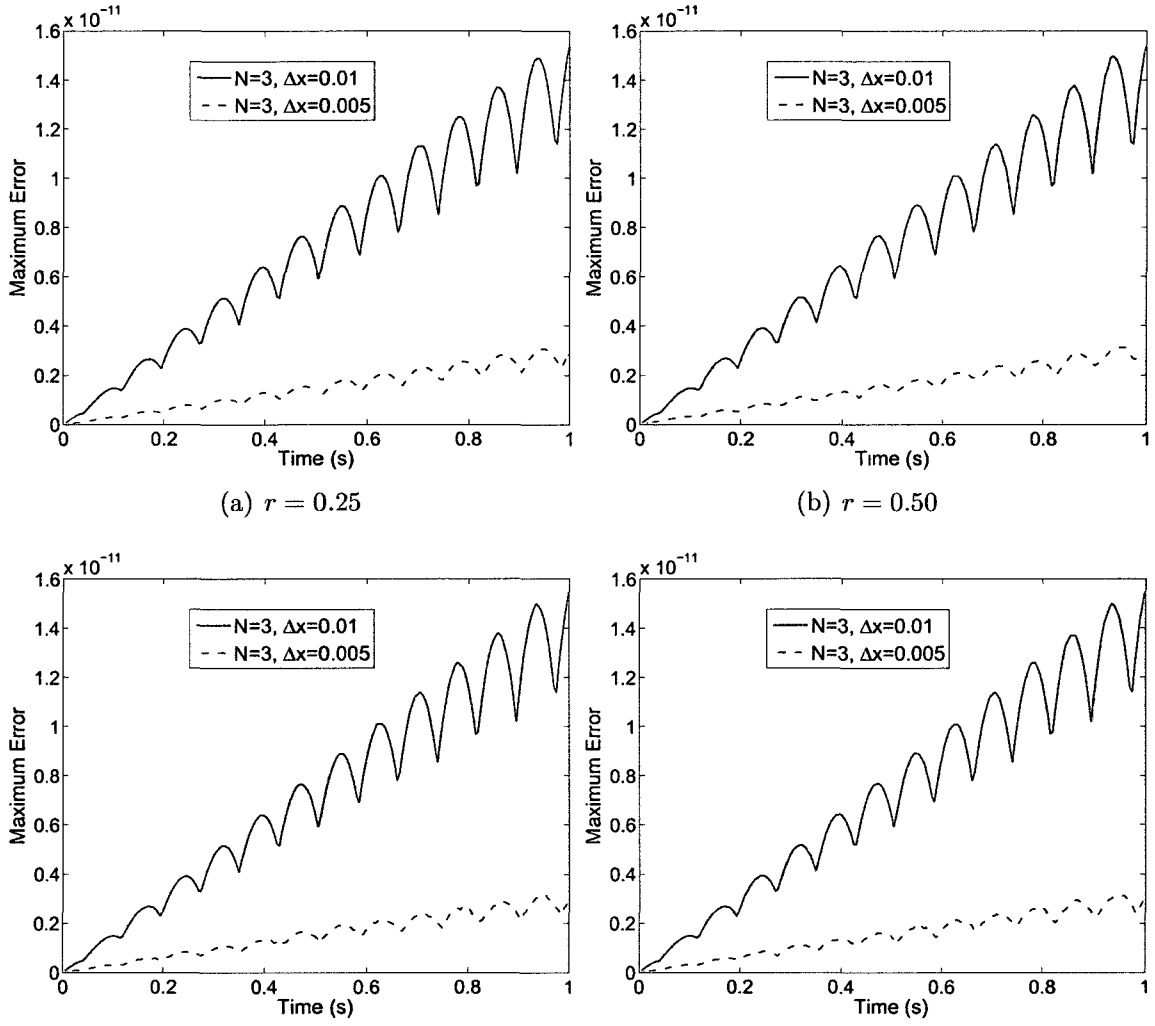
5.1.2 Numerical Results for the Model Problem

Using the computational algorithm shown in Chapter 4, numerical solutions have been obtained for each scheme at both $\Delta x = 0\ 01$ and $\Delta x = 0\ 005$, while using various mesh ratios, and choosing $N = 0$ and $N = 3$ The following graphs show the maximum absolute error for each time step plotted against time that is

$$Err^n_{\max} = \max\left\{\left|\psi^n_{\text{exact}}(x_k) - \psi^n_{\text{approx}}(x_k)\right|\right\} \qquad \text{for all } k = 0, 1, \quad , M \qquad (5\ 10)$$

where M is total number of grid points $M = \frac{1}{\Delta x}$

Figure 5 1 shows the absolute error of the model problem when choosing $\Delta x = 0\ 01$ and $\Delta x = 0\ 005$ and using four different mesh ratios Recognize that Figure 5 1(d) is slightly smaller than the critical mesh ratio for the sixth-order accurate Generalized FDTD-Q scheme from Table 5 1 which is 1 25451 for $\Delta x = 0\ 01$ and 1 25482 for $\Delta x = 0\ 005$ Choosing $r = 1\ 26$ produced a divergent result, and clearly 1 26 is larger than the critical mesh ratio indicating that the stability condition is not satisfied Specifically, $r = 1\ 26$ with $\Delta x = 0\ 01$ is equivalent to 1 02844 which is greater than one

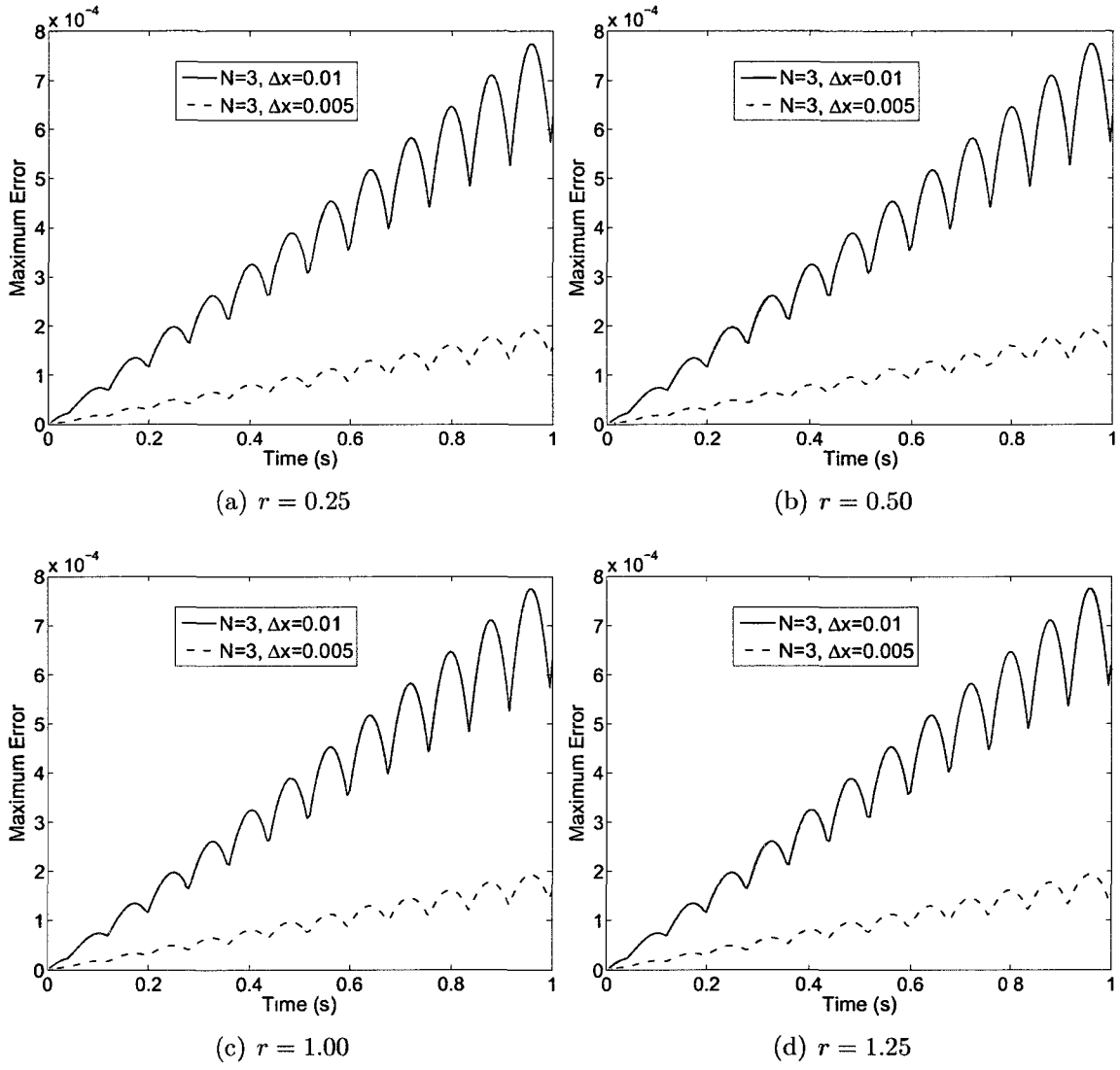


(a) $r = 0.25$ (b) $r = 0.50$

(c) $r = 1.00$ (d) $r = 1.25$

Figure 5.1: Maximum absolute error for the model problem solved with the sixth-order accurate Generalized FDTD-Q method, with $\Delta x = 0.01$ and $\Delta x = 0.005$, and $N = 3$.

Figure 5.2 contains the maximum absolute error when using the original Generalized FDTD-Q method, which used second-order accurate spatial derivatives. Note that as was published by Dai and Moxley [11], the method is convergent with $r = 1.85$, and divergent when choosing $r = 1.90$. Again from Table 5.1 it is clear that 1.90 is larger than the critical mesh ratio for the second-order accurate scheme.

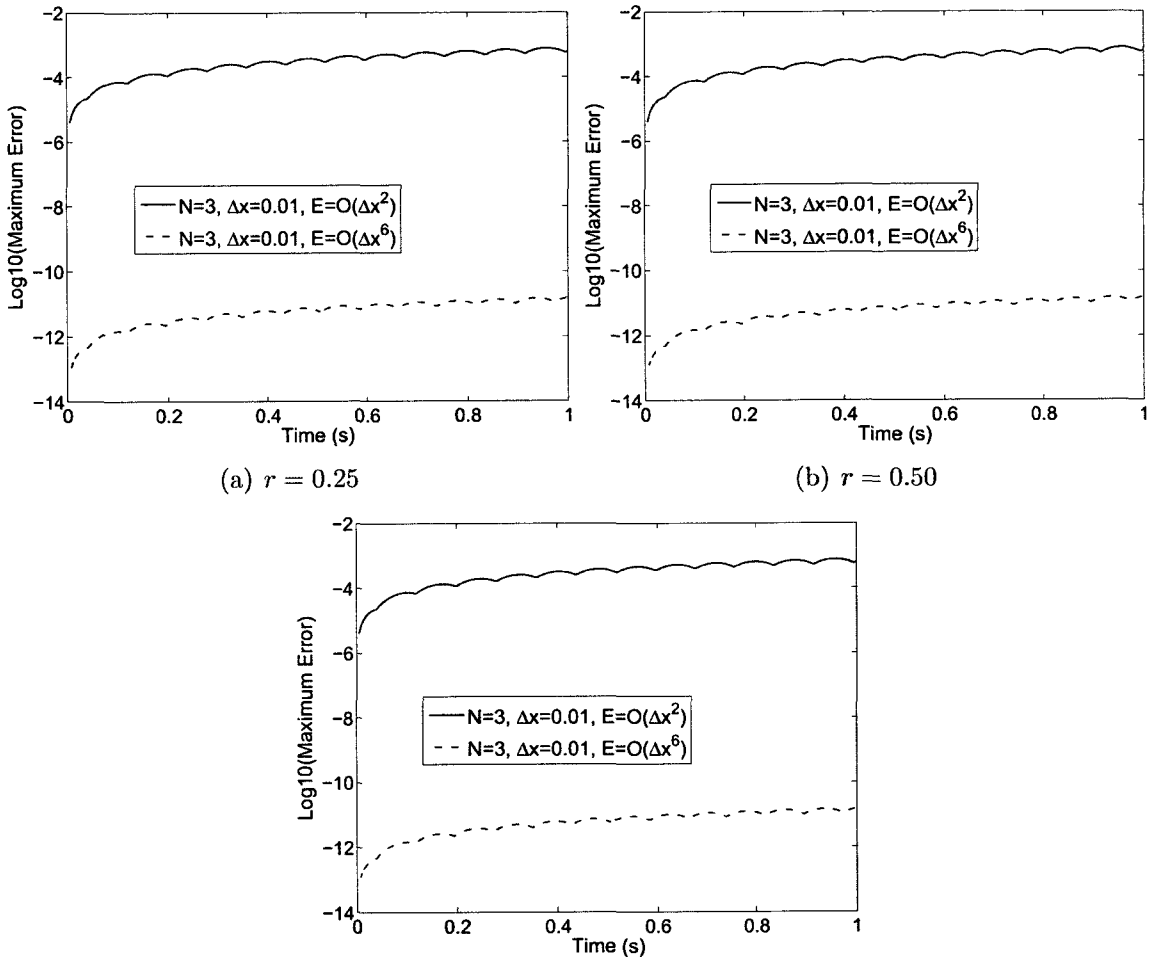


(a) $r = 0.25$ (b) $r = 0.50$

(c) $r = 1.00$ (d) $r = 1.25$

Figure 5.2: Maximum absolute error for the model problem solved with the second-order accurate Generalized FDTD-Q method, with $\Delta x = 0.01$ and $\Delta x = 0.005$, and $N = 3$.

Of note is the order of magnitude of the error in each plot, for the published Generalized FDTD-Q method the error lies in the range 0 to 8×10^{-4}, while the sixth-order accurate scheme has error in the range 0 to 1.8×10^{-11}. To summarize how the sixth-order accurate Generalized FDTD-Q scheme improves the accuracy of the solution, the $\log_{10}$ of the error is plotted against time, and is shown in Figure 5.3 when $\Delta x = 0.01$ and Figure 5.4 when $\Delta x = 0.005$.

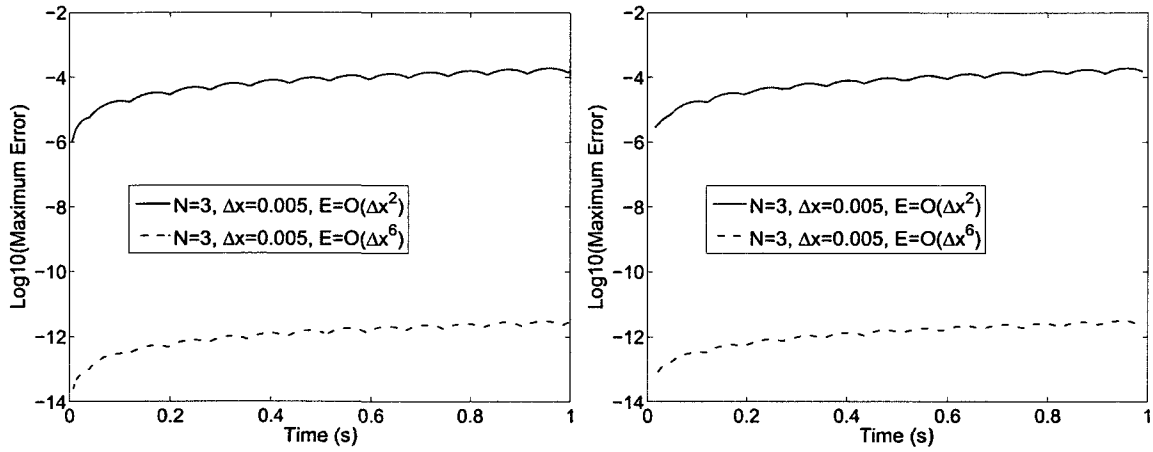


(a) $r = 0.25$

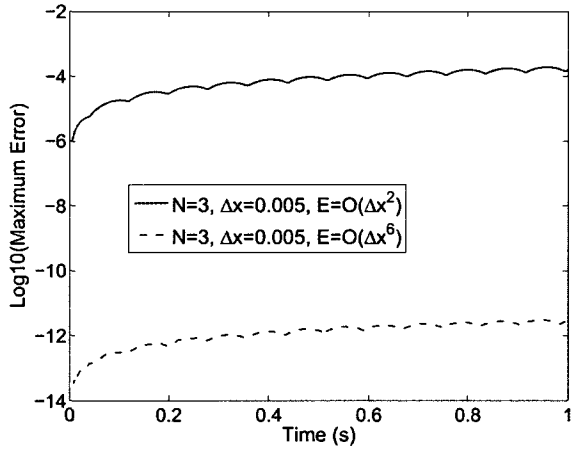
(b) $r = 0.50$

(c) $r = 1.00$

Figure 5.3: The $\log_{10}$ of the maximum absolute error for the sixth-order accurate and second-order accurate Generalized FDTD-Q methods, with $\Delta x = 0.01$ and $N = 3$.



(a) $r = 0.25$ (b) $r = 0.50$



(c) $r = 1.00$

Figure 5.4: The $\log_{10}$ of the maximum absolute error for the sixth-order accurate and second-order accurate Generalized FDTD-Q methods, with $\Delta x = 0.005$ and $N = 3$.

5.1.3 Order of Accuracy of the Spatial Derivatives

From the figures in the previous section, it is clear that the sixth-order accurate scheme shows a stark improvement in the accuracy of the solution when compared to the second-order accurate Generalized FDTD-Q method. The stability condition from Theorem 4.1 also held in all numerical tests. Next, one may wish to verify the order of accuracy of the spatial derivatives in the scheme. To do this, one must minimize the impact of the time step (Δt), because the order of accuracy of a FDTD scheme depends on both space and time. For the sixth-order accurate

Generalized FDTD-Q method, the order of accuracy was shown in Section 4 1 to be

$$O(\Delta x^6 + \Delta x^6 \Delta t^2 + \Delta x^6 \Delta t^4 + \cdot + \Delta x^6 \Delta t^{2N} + \Delta t^{2N+2}) \tag{5 11}$$

If $\Delta t \to 0$, then one will be left with only $O(\Delta x^6)$, but clearly Δt may not be zero in practice and so Δt is chosen to be very small e g , $\Delta t = 1 \times 10^{-7}$ Note that the mesh ratio r plays no part in this analysis One is effectively choosing the smallest Δt such that a solution may be computable in the time given, and Δx as large as possible such that the solution is still computable over the domain $0 \leq x \leq 1$ For example, if $\Delta x = 0\ 1$, then there are only $1/\Delta x$ grid points, and when using the sixth-order accurate central difference the three points nearest each boundary are uncomputable, meaning the method is only computing a solution on five points It was found empirically that $\Delta x = 0\ 05$ is sufficient It should also be noted that when $\Delta t = 1 \times 10^{-7}$, the solution over the interval $0 < t \leq 1$ requires to 10 000,000 time steps

The order of accuracy is approximated as follows, one computes the solution and associated absolute error using $\Delta t = 1 \times 10^{-7}$ and $\Delta x = 0\ 05$, and also the solution and absolute error using $\Delta t = 1 \times 10^{-7}$ and $\frac{\Delta x}{2} = 0\ 025$ The error for each may then be defined as

$$Err_{\Delta x} = O(\Delta x^n), \tag{5 12}$$

$$Err_{\frac{\Delta x}{2}} = O\Big((\frac{\Delta x}{2})^n\Big) = O(\Delta x^n) \quad \frac{1}{2^n} \tag{5 13}$$

Substituting Equation (5 12) into Equation (5 13) allows one to remove the unknown terms in the truncation error e g $C \quad f^{(8)}(\xi)$, yielding

$$Err_{\frac{\Delta x}{2}} = Err_{\Delta x} \quad \frac{1}{2^n}$$

$$2^n = \frac{Err_{\Delta x}}{Err_{\frac{\Delta x}{2}}}$$

$$n = \log\left(\frac{Err_{\Delta x}}{Err_{\frac{\Delta x}{2}}}\right) \cdot \frac{1}{\log(2)}. \tag{5.14}$$

Using the above method to compute the approximate order of accuracy of the spatial derivative, Figure 5.5 shows the graph of the order of accuracy of the spatial derivative over time. One can see that the method has approximately sixth-order accuracy at the beginning of the time interval, and slowly over time it degrades. Using this method to approximate the order of accuracy of the spatial derivatives, the minimum, maximum, and mean approximate orders of accuracy are shown in Table 5.2, for problems solved using both the sixth-order and second-order accurate Generalized FDTD-Q methods, and choosing N to be zero and three.

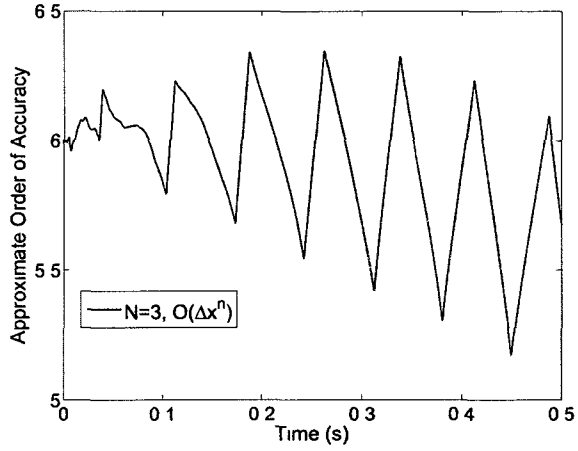


Figure 5.5: The observed order of accuracy of the spatial derivative when using the sixth-order accurate Generalized FDTD-Q method computed using Equation (5.14), with $\Delta x = 0.05$ and $\Delta x = 0.025$, $\Delta t = 1 \times 10^{-7}$, and $N = 3$.

Table 5.2: Approximate order of accuracy of the Generalized FDTD-Q method using second-order and sixth-order accurate spatial derivatives, and $\Delta x = 0.05$ and $\Delta x = 0.025$, and $\Delta t = 1 \times 10^{-7}$.

Approximate Order of Accuracy for the Spatial Derivative						
	$O(\Delta x^2)$ Scheme			$O(\Delta x^6)$ Scheme		
N	min	max	mean	min	max	mean
0	1.9943	2.0039	1.991	5.1722	6.3473	5.9099
3	1.9943	2.0039	1.991	5.1722	6.3473	5.9099

5.1.4 Conclusions

Having shown theoretical guidelines for using the sixth-order accurate central differences in the Generalized FDTD-Q method in Section 4.2 and arrived at Theorem 4.1; the numerical results presented for the model problem behaved precisely in line with the theoretical basis. The critical mesh ratios shown in Table 5.1 were constructed from theory, yet clearly observed in Figure 5.2 and Figure 5.1. Furthermore, the magnitude of the error decreased drastically when compared against the second-order accurate scheme as shown in Figure 5.3 and Figure 5.4, and the observed order of accuracy of the spatial derivatives was maintained for over $5,000,000$ time steps as shown in Table 5.2. Based on these observations as well as the theoretical foundation presented in Chapter 4, it has been shown that the sixth-order accurate central differences have shown a measurable improvement over the second-order accurate Generalized FDTD-Q scheme when applied to the model problem.

5.2 Particle Simulation

Having used the model problem to bridge the gap between theory and computation, and in doing so verified that the theoretical underpinnings presented in this writing. Another problem is solved in this section with the motivation being to show the practical value of the Generalized FDTD-Q method. Following in the footsteps of Sullivan [29] and Dai [10, 11], the Generalized FDTD-Q scheme will be used to simulate a particle moving in 1-D free space and then hitting an energy potential. The following problem is taken directly from Sullivan's book [29], a particle is initiated at a wavelength of λ in a Gaussian envelop of width σ with the following two equations:

$$\psi^0_{\text{real}}(k) = e^{-0.5(\frac{k-k_0}{\sigma})^2} \cos\left(\frac{2\pi(k-k_0)}{\lambda}\right) \tag{5.15a}$$

and

$$\psi^0_{\text{imag}}(k) = e^{-0.5(\frac{k-k_0}{\sigma})^2} \sin\left(\frac{2\pi(k-k_0)}{\lambda}\right), \tag{5.15b}$$

where k_0 is the center of the pulse.

The specific grid size chosen will be chosen from Dai [10], and a mesh of 1600 spatial grid points is constructed with k_0= 400 and $\sigma = \lambda = 1.0 \times 10^{-10}$ [m]. The parameters required by the Schrödinger equation Equation (1.1) are defined by the simulation itself, for this simulation we seek to model an electron moving through 1-D free-space, and therefore m is taken to be the mass of an electron, Δx is chosen to be one-tenth of an Angstrom, and $\hbar$ is the reduced Planck's constant

$$m = 9.1 \times 10^{-31} \text{ [kg]} \tag{5.16}$$

$$\Delta x = 1.0 \times 10^{-11} \text{ [m]} \tag{5.17}$$

$$\hbar = 1.054 \times 10^{-34} \text{ [J} \cdot \text{sec]}. \tag{5.18}$$

To replicate Dai's results, V was chosen to be 0 in the first 800 grid points and 100 [eV] in the next 800 grid points. To have the units match, V must be expressed in Joules, and so the conversion

$$1 \text{ [eV]} = 1.602 \times 10^{-19} \text{ [J]} \tag{5.19}$$

will be used when necessary.

The next equations are again taken from Sullivan [29], and are used to determine the expected energy, both Kinetic and Potential that should exist in the system. They are computed from $\psi^n_{\text{real}}(k)$ and $\psi^{n+\frac{1}{2}}_{\text{imag}}(k)$ in the simulation as follows :

$$\text{Kinetic Energy (KE)} = -\frac{\hbar^2}{2m} \sum_{k=1}^{N} \left[\psi^n_{\text{real}}(k) - \imath\psi^{n+\frac{1}{2}}_{\text{imag}}(k)\right] \cdot \left[\frac{\partial^2 \psi^n_{\text{real}}(k)}{\partial x^2} + \imath \frac{\partial^2 \psi^{n+\frac{1}{2}}_{\text{imag}}(k)}{\partial x^2}\right] \tag{5.20}$$

and

$$\text{Potential Energy (PE)} = \sum_{k=1}^{N} V(k) \quad \left[[\psi^{n}_{\text{real}}(k)]^2 + [\psi^{n+\frac{1}{2}}_{\text{imag}}(k)]^2 \right] \tag{5 21}$$

The approximation of the Laplace operator was chosen to be a sixth-order accurate (7-point) central difference approximation

$$\begin{aligned} \frac{\partial^2 \psi^{n}_{\text{real}}(k)}{\partial x^2} \approx \frac{1}{180\Delta x^2} \Big[& 2\psi^{n}_{\text{real}}(k+3) - 27\psi^{n}_{\text{real}}(k+2) + 270\psi^{n}_{\text{real}}(k+1) \\ & - 490\psi^{n}_{\text{real}}(k) + 270\psi^{n}_{\text{real}}(k-1) - 27\psi^{n}_{\text{real}}(k-2) \\ & + 2\psi^{n}_{\text{real}}(k-3) \Big] \end{aligned} \tag{5 22a}$$

and

$$\begin{aligned} \frac{\partial^2 \psi^{n+\frac{1}{2}}_{\text{imag}}(k)}{\partial x^2} \approx \frac{1}{180\Delta x^2} \Big[& 2\psi^{n}_{\text{imag}}(k+3) - 27\psi^{n}_{\text{imag}}(k+2) + 270\psi^{n}_{\text{imag}}(k+1) \\ & - 490\psi^{n}_{\text{imag}}(k) + 270\psi^{n}_{\text{imag}}(k-1) - 27\psi^{n}_{\text{imag}}(k-2) \\ & + 2\psi^{n}_{\text{imag}}(k-3) \Big] \end{aligned} \tag{5 22b}$$

From Dai [10] the simulation should model an electron moving through free space and then hitting an energy potential with a total of about 150 (eV) The energy is purely kinetic due to the fact that there is no potential energy available before the energy barrier is reached With an increase in time, the electron will propagate in the positive spatial direction The waveform begins to spread, but the total kinetic energy remains constant After the electron strikes the potential barrier, part of the energy will be converted to potential energy The waveform indicates that there is some probability that the electron is reflected and some probability that it penetrates the potential barrier However the total energy should remain constant

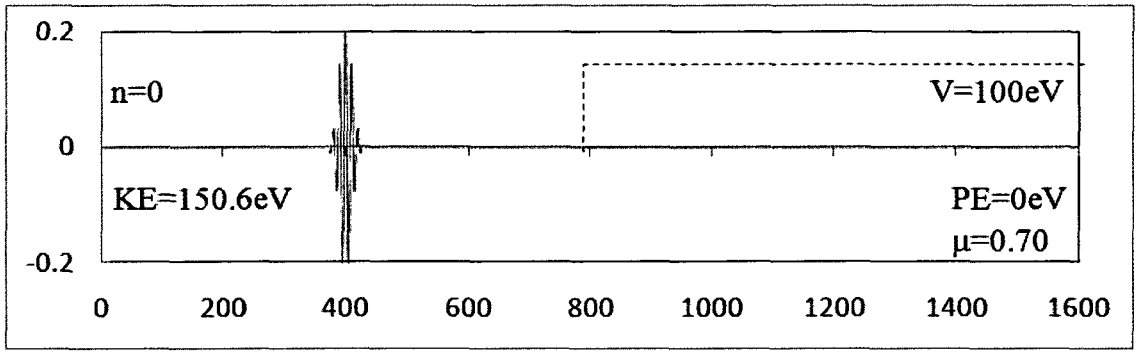
The important aspect that this simulation will evaluate is the last statement taken from Dai, that is the total energy should remain constant Compared against Dai and Moxley [11], the plots will evaluate the impact of improving the accuracy

of the approximation of the Laplace operator in the Generalized FDTD-Q scheme Table 5 3 shows the findings published with the Generalized FDTD-Q method, which utilized second-order accurate spatial derivatives, and a second-order accurate central difference approximation of the Laplace operator used to compute the Kinetic energy

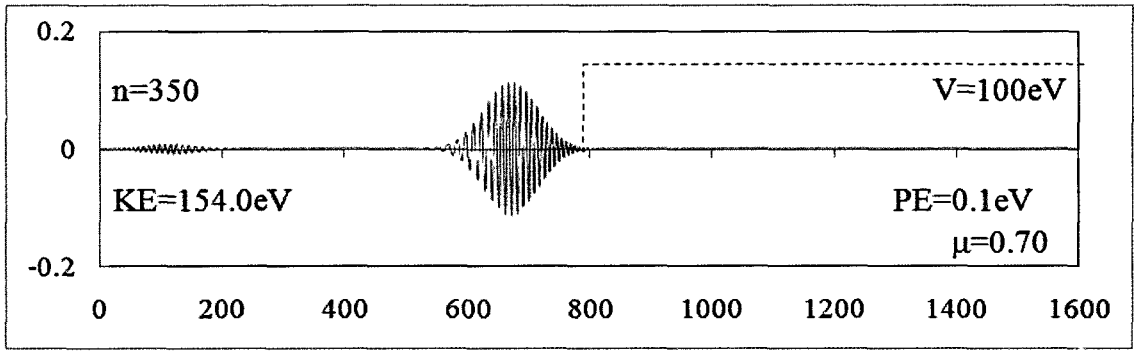
Table 5 3 Energy conservation of Generalized FDTD-Q method with second-order and sixth-order accurate spatial derivatives

Energy Conservation of the Generalized FDTD-Q Schemes						
	$O(\Delta x^2)$ Scheme [11]			$O(\Delta x^6)$ Scheme		
N	0	350	1300	0	350	1300
0 70	151	154	149	151	154	152
0 75	151	154	0	151	154	152
0 95	___	___	___	151	151	154
1 00	___	___	___	151	162	196

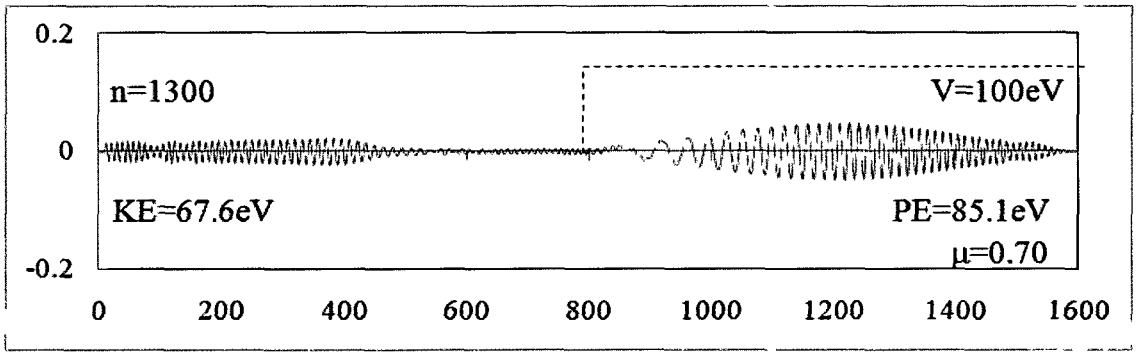
One can see from Table 5 3 that as the stability condition is relaxed, the energy conservation becomes increasingly poor Figures 5 6, 5 7, 5 8, and 5 9 show the particle simulations with the stability condition μ at $\mu = 0\ 70$, $\mu = 0\ 75$, $\mu = 0\ 95$, and $\mu = 1\ 00$ While the plots for $\mu = 0\ 95$ and $\mu = 1\ 00$ are not too different one can clearly see from Table 5 3 that the energy conservation for $\mu = 1\ 00$ is becoming highly inaccurate



(a) Time t=0

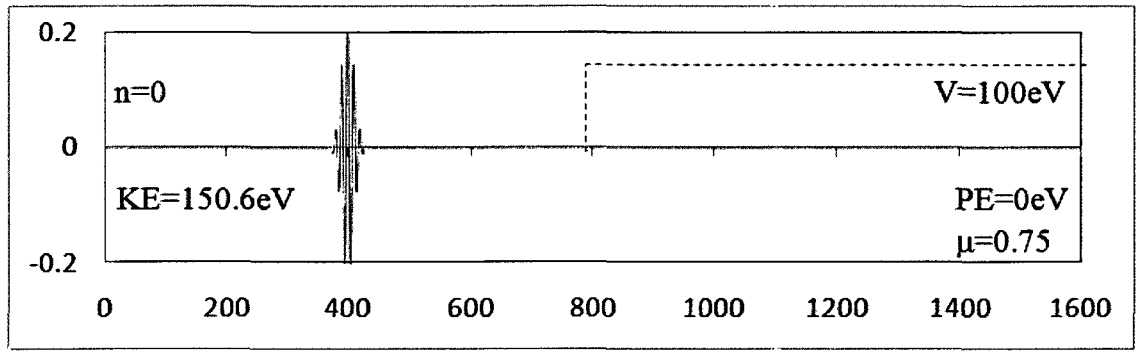


(b) Time n=350

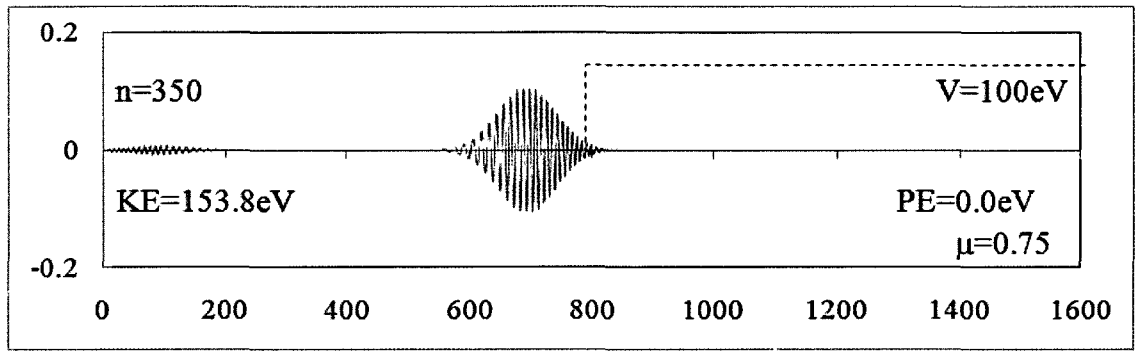


(c) Time n=1300

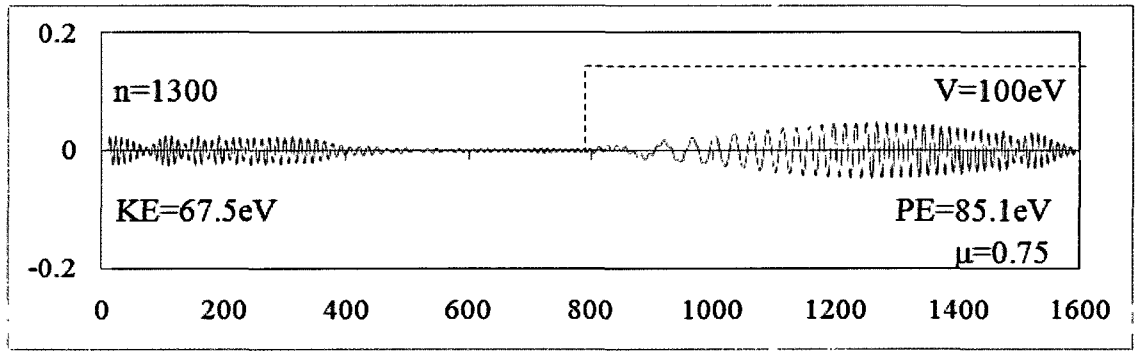
Figure 5.6: Particle simulation using stability condition $c = 0.70$.



(a) Time t=0

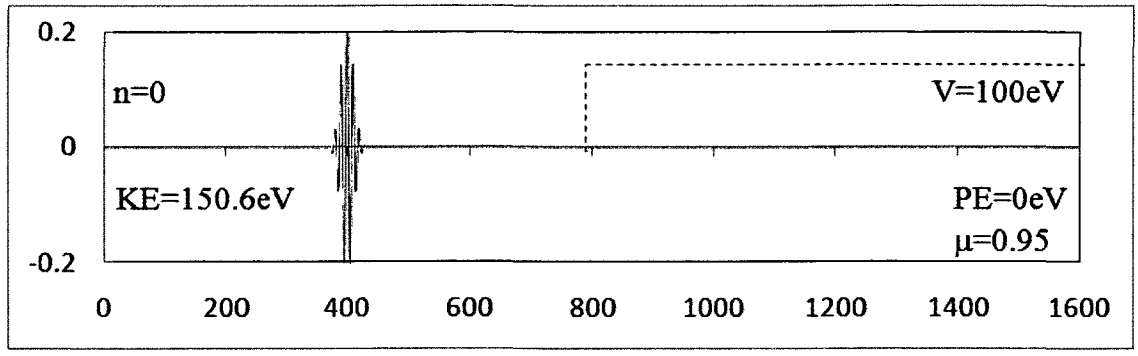


(b) Time n=350

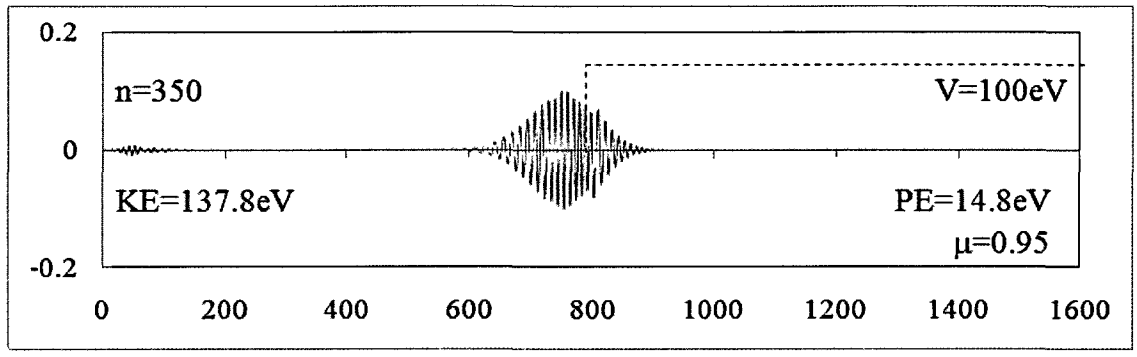


(c) Time n=1300

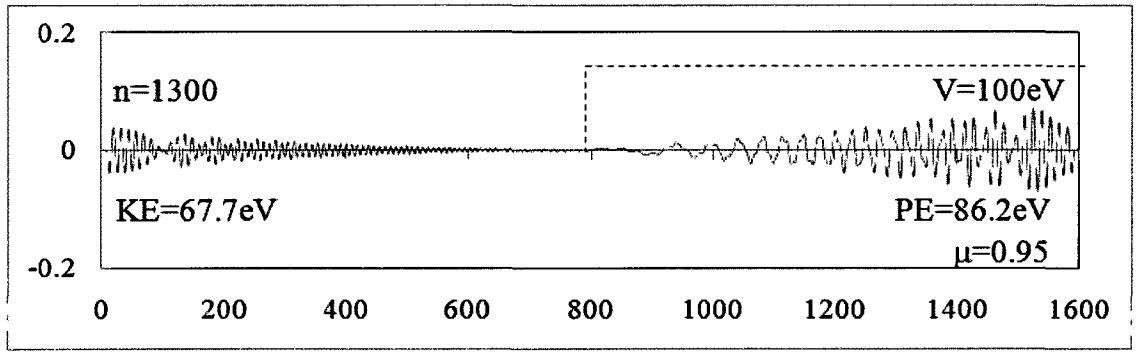
Figure 5.7: Particle simulation using stability condition $c = 0.75$.



(a) Time t=0

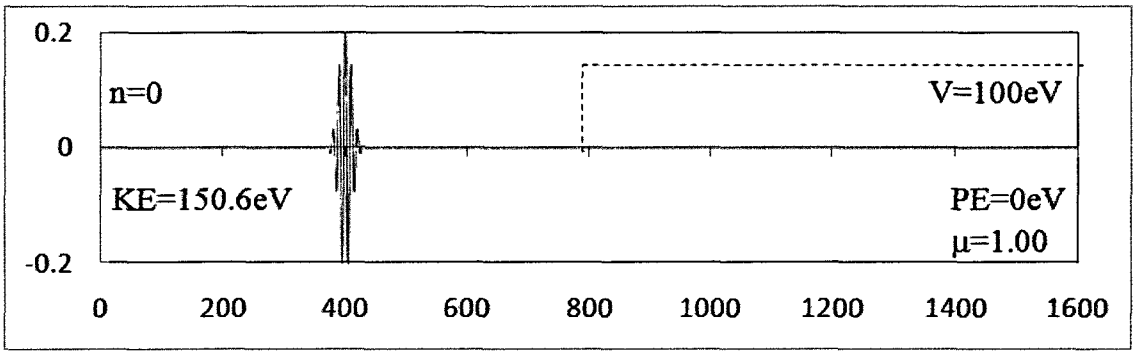


(b) Time n=350

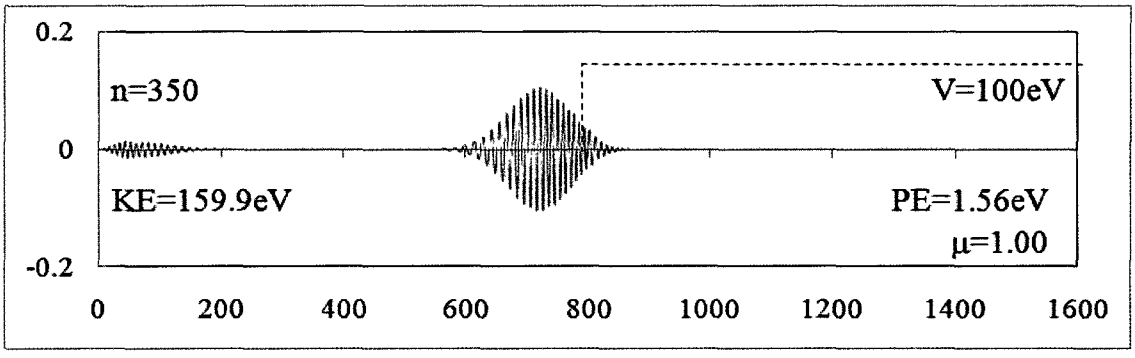


(c) Time n=1300

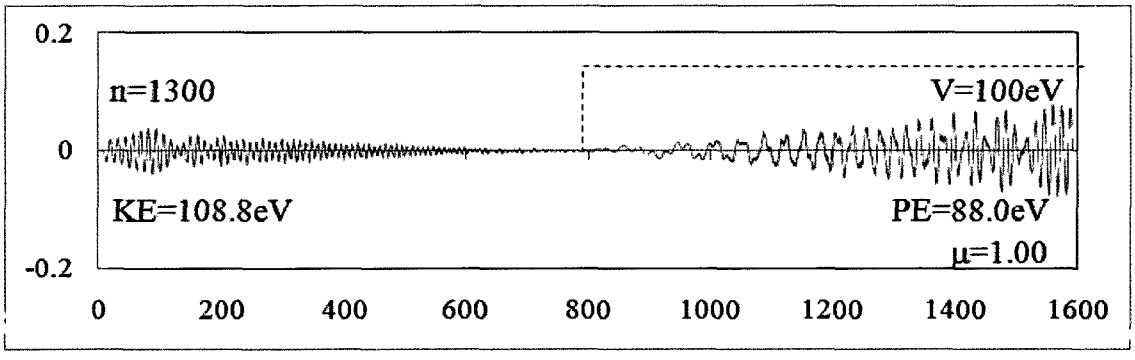
Figure 5.8: Particle simulation using stability condition $c = 0.95$.



(a) Time t=0



(b) Time n=350



(c) Time n=1300

Figure 5.9: Particle simulation using stability condition $c = 1.00$.

5.3 Conclusions and Future Work

The work conducted in this dissertation has shown a measurable benefit for the Generalized FDTD-Q method, as is evident from the solutions to the model problem. In the simulations of the particle moving in free space, it is also clear that numerical feedback (reflections of the wave off the boundaries) is present. These small oscillations likely pollute the solution and should be damped using an Absorbing Boundary Condition. From Table 5.3 it is not clear if increasing the accuracy of the spatial derivatives provides any benefit in regard to conserving energy, though it is hypothesized that with a suitable absorbing boundary condition, the impact of higher accuracy spatial derivatives will become more apparent.

We would also like to revisit the use of piecewise low degree Lagrange interpolating polynomials, possibly as tools to aid in providing information at the uncomputable points when using central differences. We would also like to experiment with the use of Richardson extrapolation to improve the accuracy of the higher-order derivatives, e.g., sixth, eighth, and higher-order derivatives. Ideally, we would like to improve the accuracy of the accuracy of the Generalized FDTD-Q scheme such that the order of accuracy has the form $O(\Delta x^2 + \Delta x^2\Delta t^2 + \Delta x^4\Delta t^4 + \cdots + \Delta x^{2N}\Delta t^{2N} + \Delta t^{2N+2})$, doing so the Author hypothesizes that an N of two or three may be sufficient.

APPENDIX A

SAMPLE ERROR PLOTS

Contained in this appendix are additional error plots for the test function $f(x) = e^{(-\frac{x^2}{2})}$, which was introduced in Subsection 3.3.1. These error plots illustrate how the differentiated Lagrange interpolating polynomials lose accuracy as they are repeatedly differentiated. The intent of these plots is to provide a visual progression of the error propagation as successively high-order derivatives are computed. By observing the first-, second-, third-, fourth-, fifth-, and sixth-order derivatives, one can clearly see the error grow at the endpoints. The error is then plotted next to these differentiated functions, so one may see the distribution of error throughout the interval.

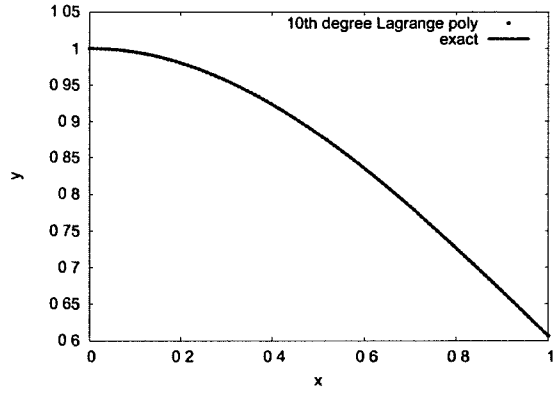
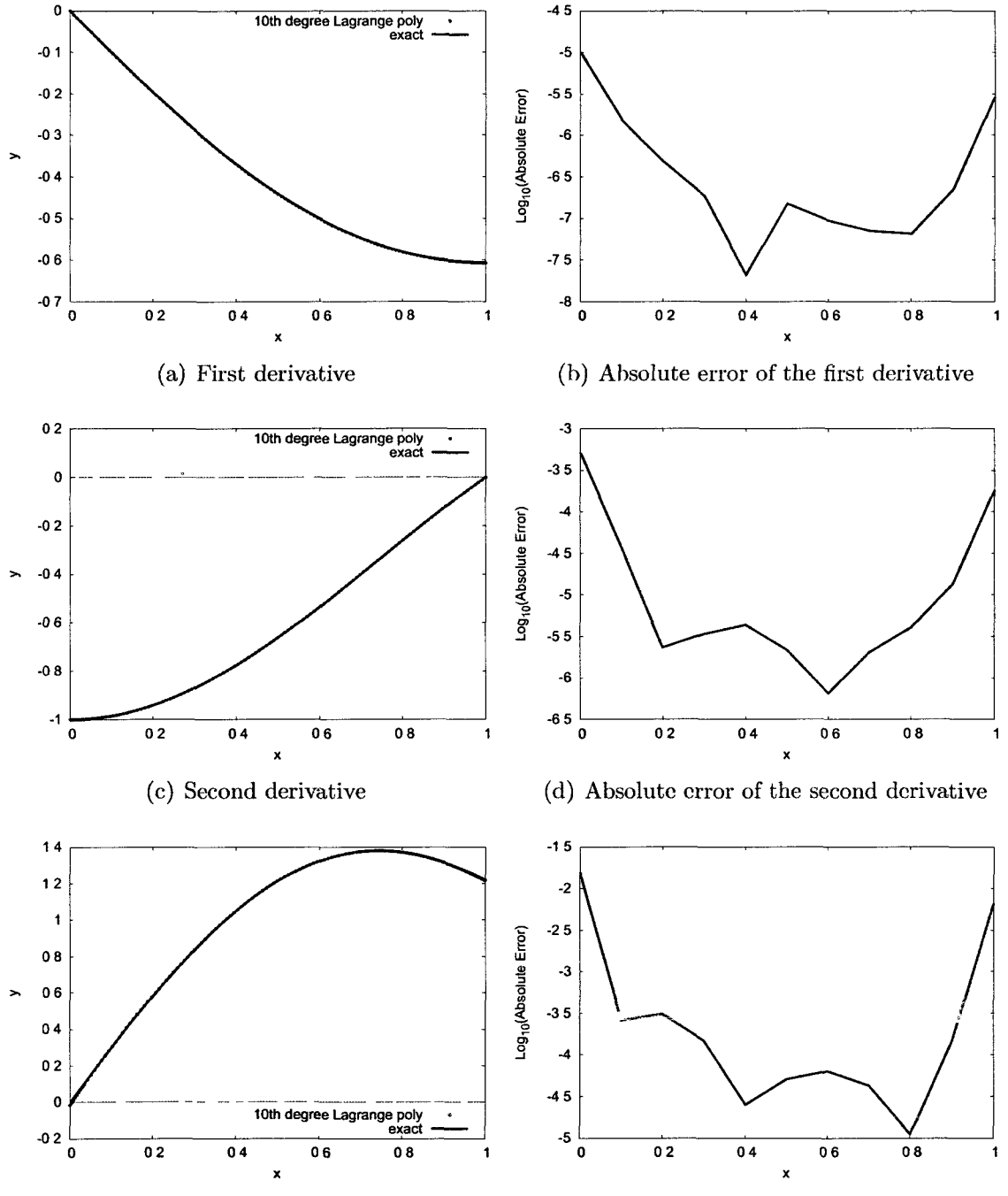


Figure A.1: Test function $f(x) = e^{(-\frac{x^2}{2})}$ plotted over the interval $[0, 1.035]$.



(a) First derivative

(b) Absolute error of the first derivative

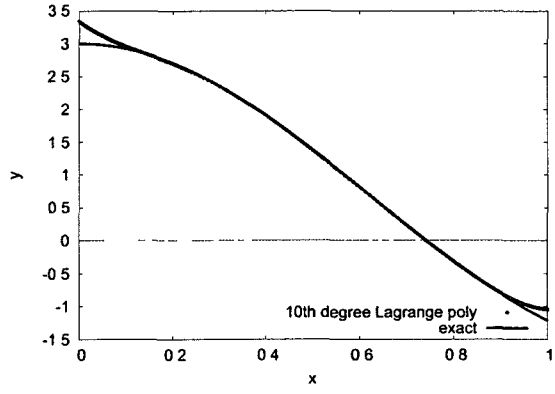
(c) Second derivative

(d) Absolute error of the second derivative

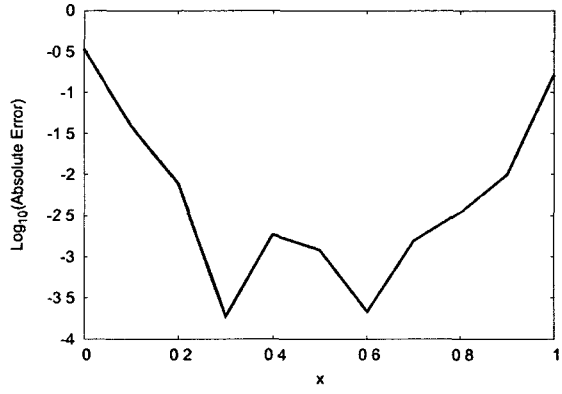
(e) Third derivative

(f) Absolute error of the third derivative

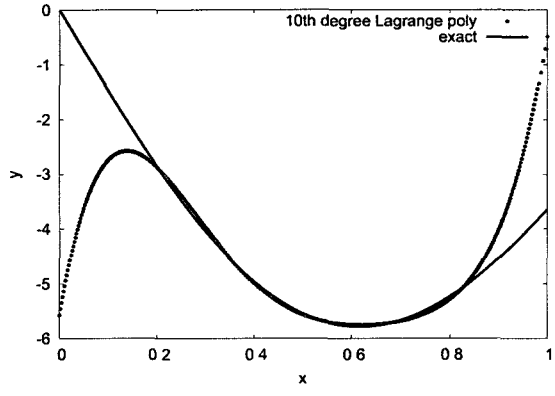
Figure A.2: First-, second-, and third-order derivatives of the test function $f(x) = e^{(-\frac{x^2}{2})}$ and associated absolute error over the interval $[0, 1.035]$.



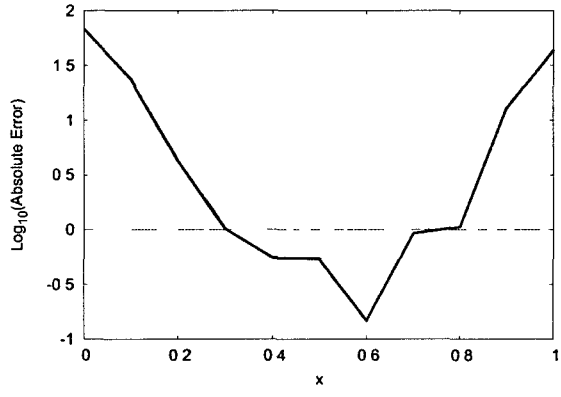
(a) Fourth derivative



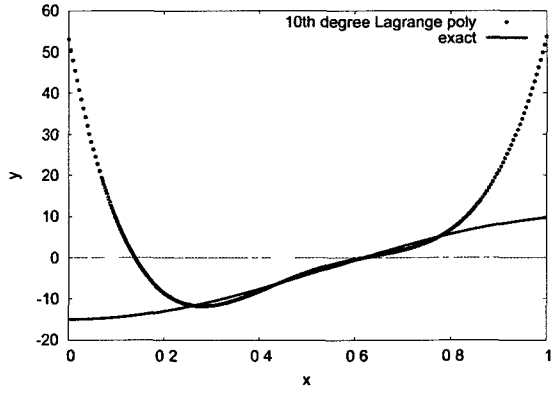
(b) Absolute error of the sixth derivative



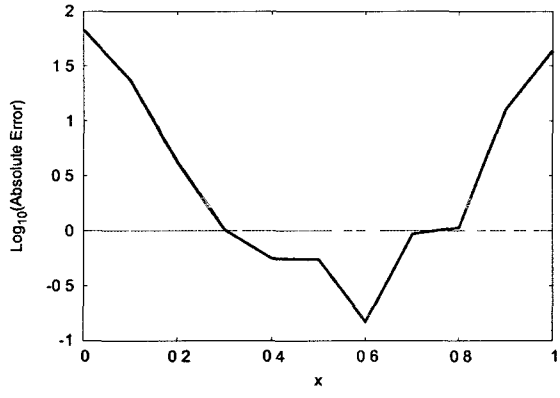
(c) Fifth derivative



(d) Absolute error of the sixth derivative



(e) Sixth derivative



(f) Absolute error of the sixth derivative

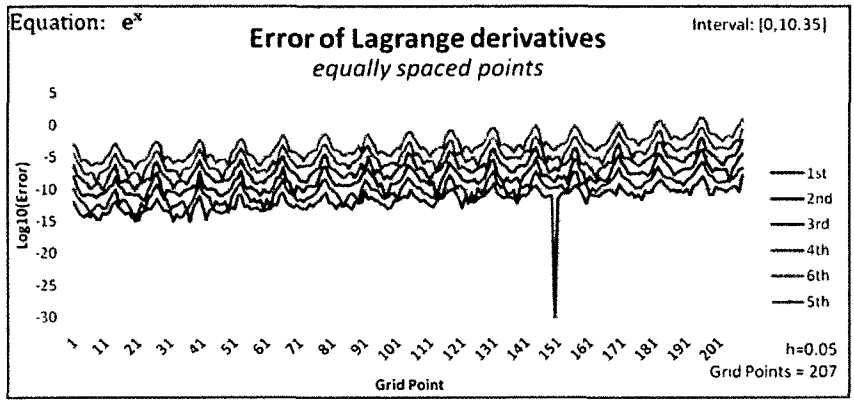
Figure A.3: Fourth-, fifth-, and sixth-order derivatives of the test function $f(x) = e^{(-\frac{x^2}{2})}$ and associated absolute error over the interval $[0, 1.035]$.

APPENDIX B

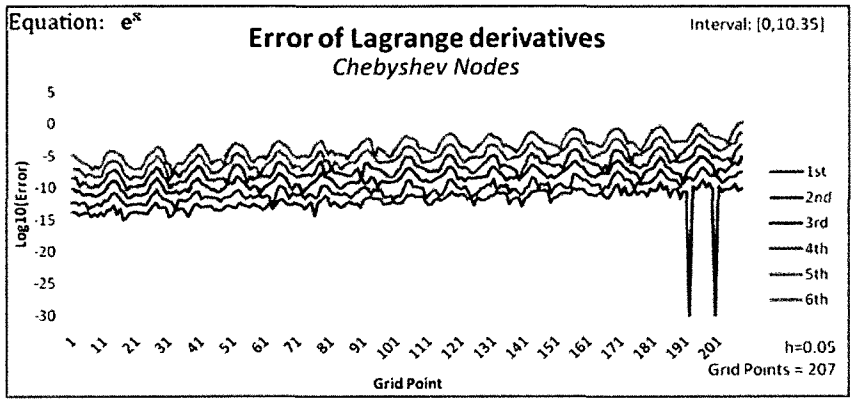
NUMERICAL DERIVATIVE RESULTS

This appendix contains additional numerical results used in comparing the differentiated Lagrange interpolating polynomials against the central difference approximations of the Laplace operator. Each section contains results using the differentiated Lagrange interpolating polynomials with three different abscissas, as well as results using the central difference approximation of the Laplace operator with three different orders of accuracy: second-order accurate, fourth-order accurate, and sixth-order accurate. For each function tested, solutions are constructed over two intervals $[0, 1.035]$ and $[0, 10.35]$.

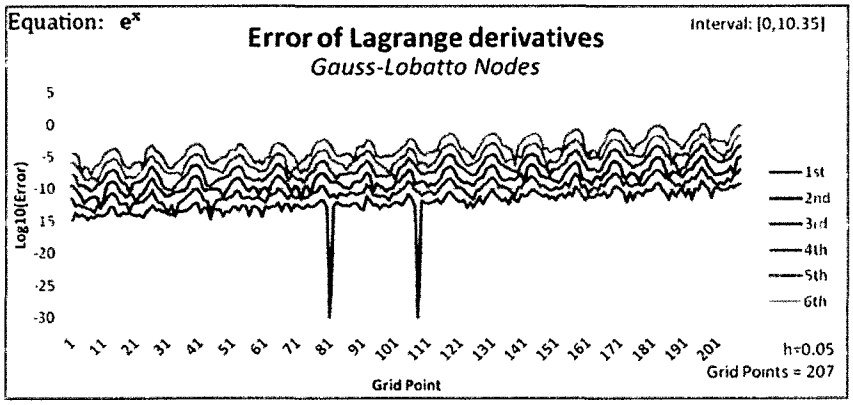
B.1 Exponential Function



(a) Differentiated Lagrange interpolating polynomials using equally spaced nodes to compute the first- through sixth-order derivatives.

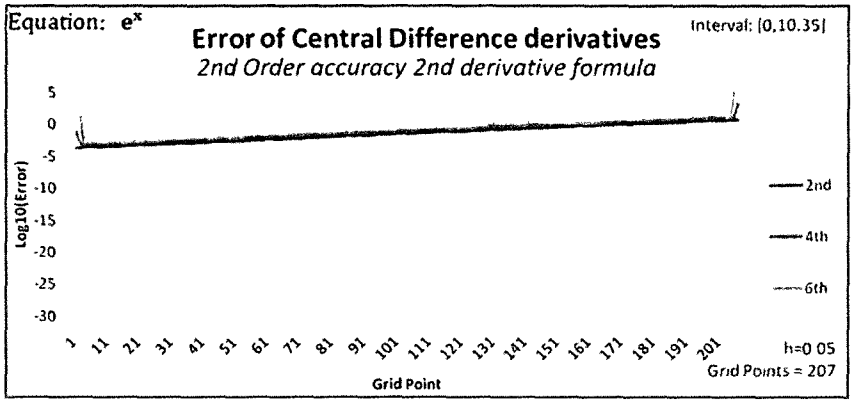


(b) Differentiated Lagrange interpolating polynomials using the Chebyshev nodes to compute the first- through sixth-order derivatives.

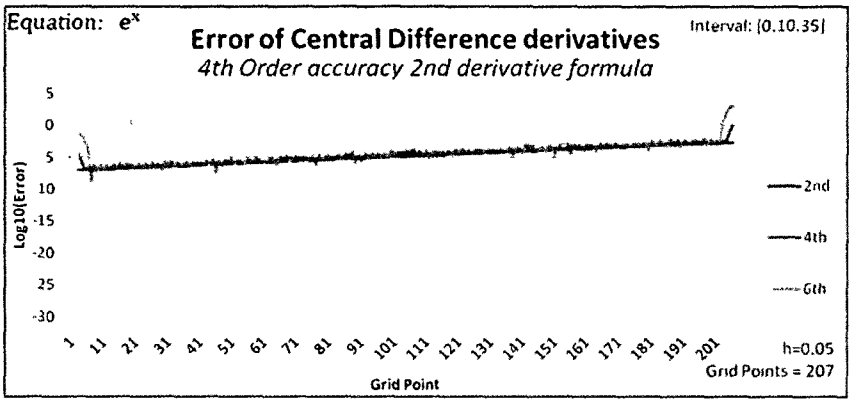


(c) Differentiated Lagrange interpolating polynomials using the Gauss-Lobatto nodes to compute the first- through sixth-order derivatives.

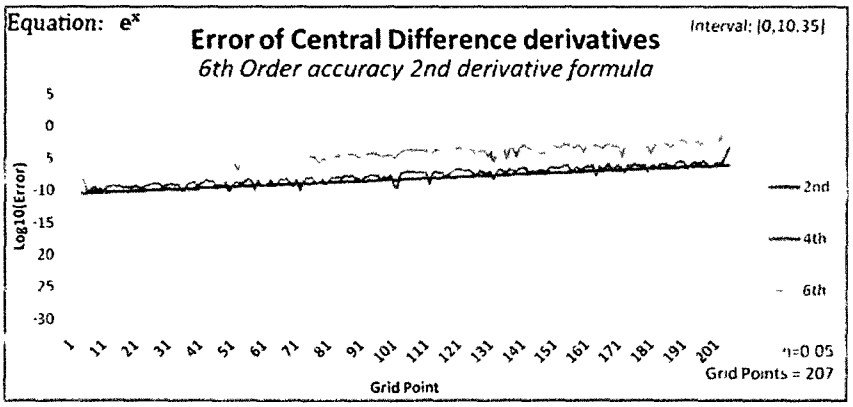
Figure B.1: Differentiated piecewise twelfth degree Lagrange interpolating polynomials for the function $f(x) = e^{(x)}$, over the interval $[0, 10.35]$, utilizing 208 total grid points.



(a) Second-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

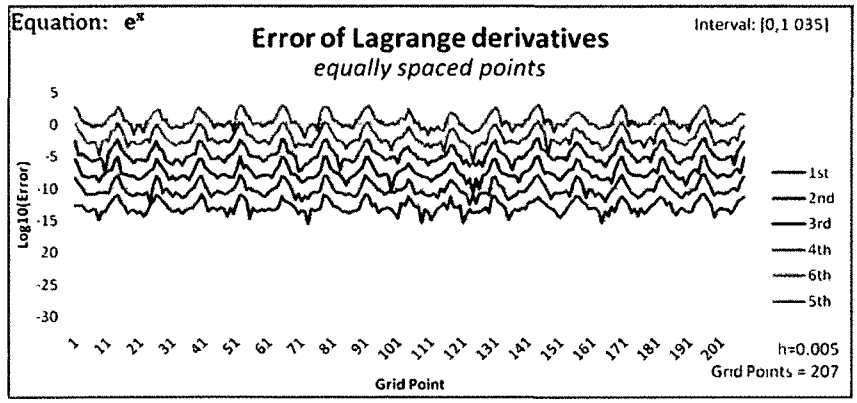


(b) Fourth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

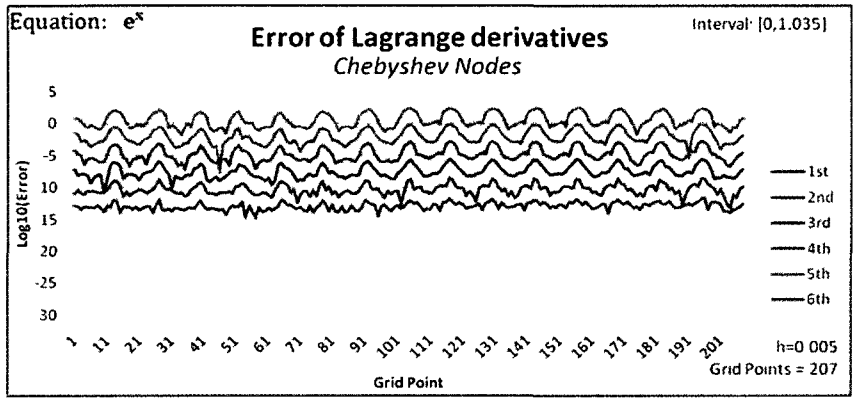


(c) Sixth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

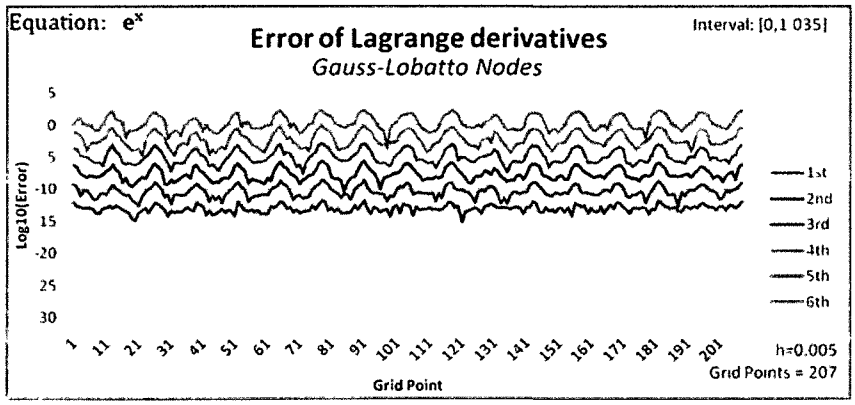
Figure B.2: Central difference approximations of the Laplace operator applied to the function $f(x) = e^{(x)}$, over the interval $[0, 10.35]$, utilizing 208 total grid points, and various orders of accuracy.



(a) Differentiated Lagrange interpolating polynomials using equally spaced nodes to compute the first- through sixth-order derivatives.

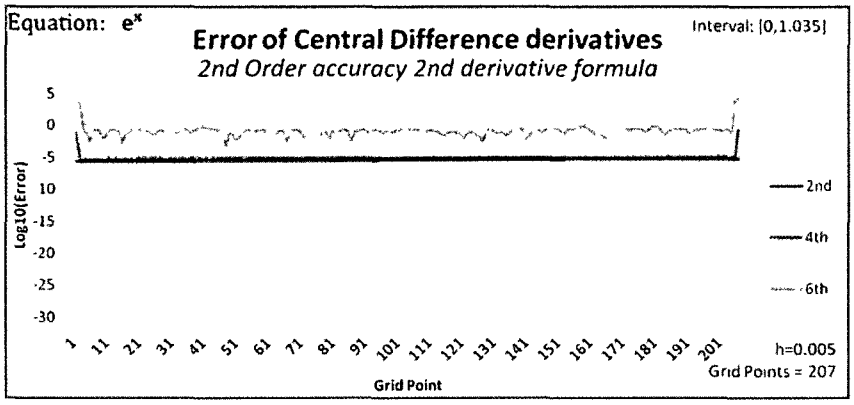


(b) Differentiated Lagrange interpolating polynomials using the Chebyshev nodes to compute the first- through sixth-order derivatives.

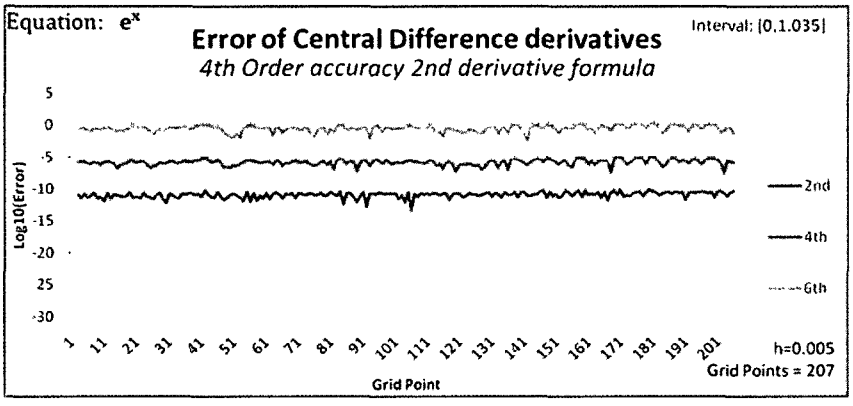


(c) Differentiated Lagrange interpolating polynomials using the Gauss-Lobatto nodes to compute the first- through sixth-order derivatives.

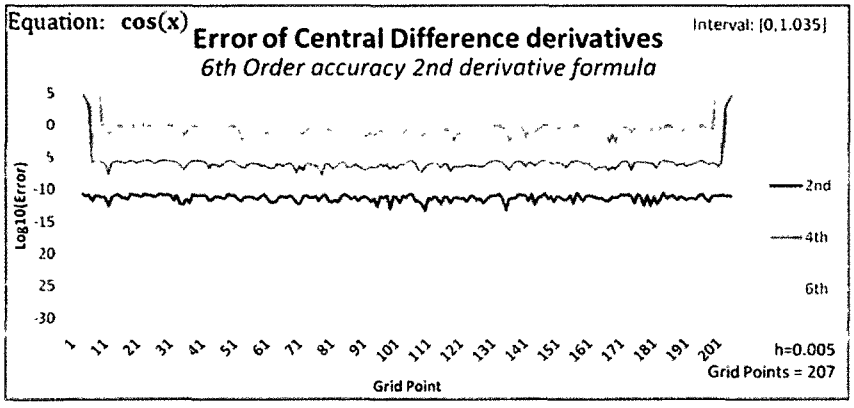
Figure B.3: Differentiated piecewise twelfth degree Lagrange interpolating polynomials for the function $f(x) = e^{(x)}$, over the interval $[0, 1.035]$, utilizing 208 total grid points.



(a) Second-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.



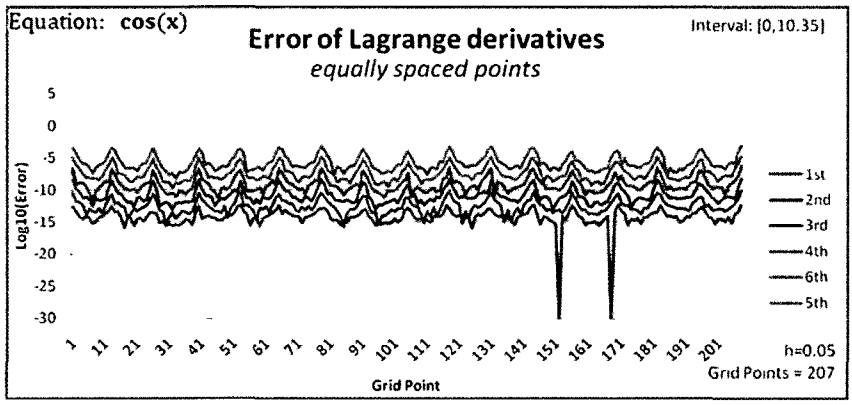
(b) Fourth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.



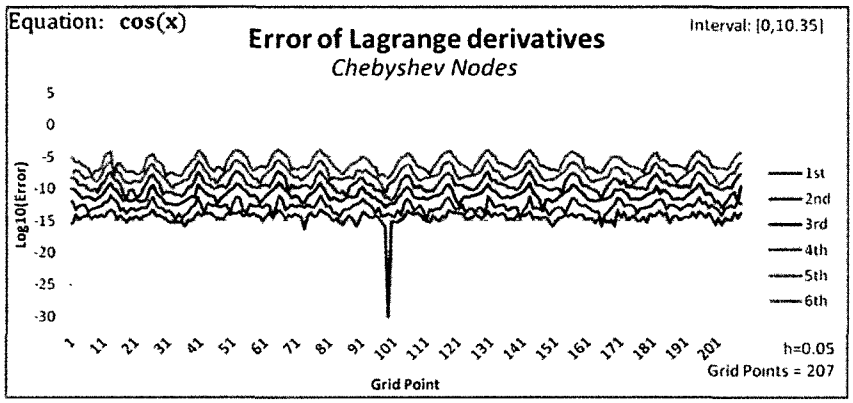
(c) Sixth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

Figure B.4: Central difference approximations of the Laplace operator applied to the function $f(x) = e^{(x)}$, over the interval $[0, 1.035]$, utilizing 208 total grid points, and various orders of accuracy.

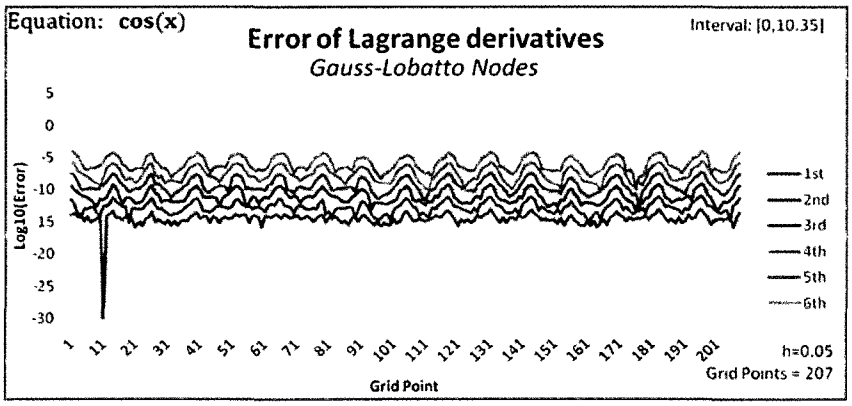
B.2 Trigonometric Function



(a) Differentiated Lagrange interpolating polynomials using equally spaced nodes to compute the first- through sixth-order derivatives.

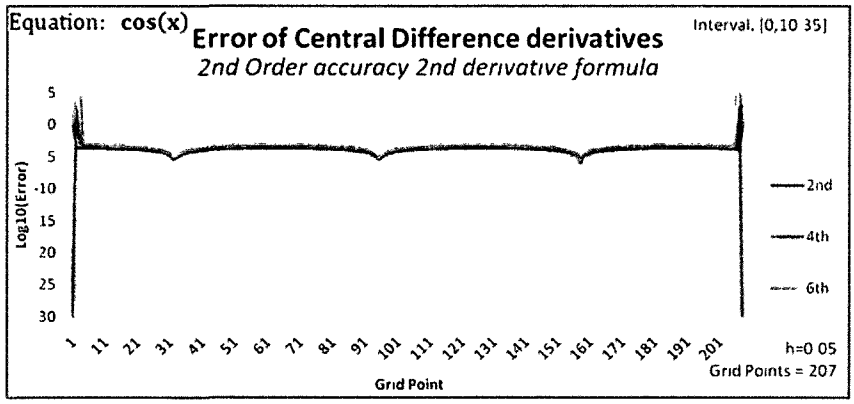


(b) Differentiated Lagrange interpolating polynomials using the Chebyshev nodes to compute the first- through sixth-order derivatives.

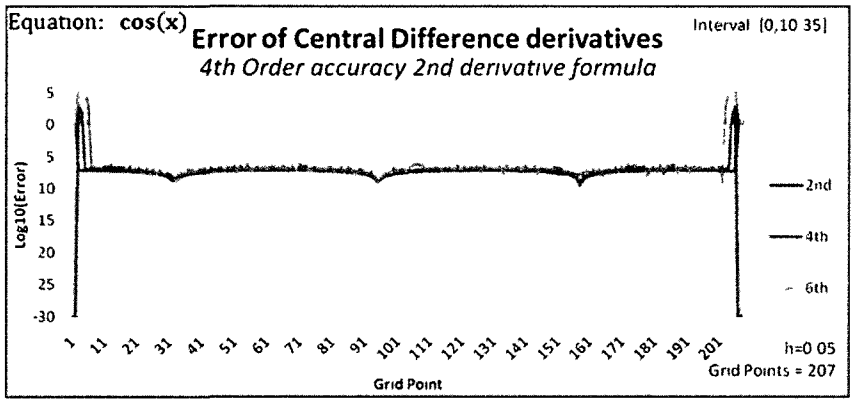


(c) Differentiated Lagrange interpolating polynomials using the Gauss-Lobatto nodes to compute the first- through sixth-order derivatives.

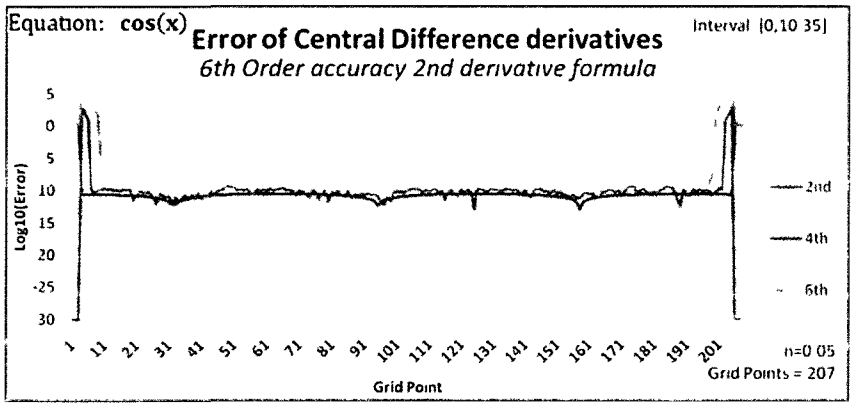
Figure B.5: Differentiated piecewise twelfth degree Lagrange interpolating polynomials for the function $f(x) = \cos(x)$, over the interval $[0, 10.35]$, utilizing 208 total grid points, and three different grid spacings.



(a) Second-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

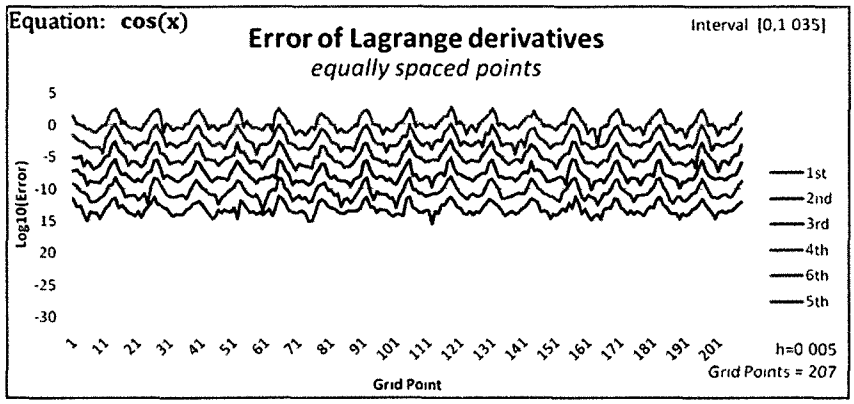


(b) Fourth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

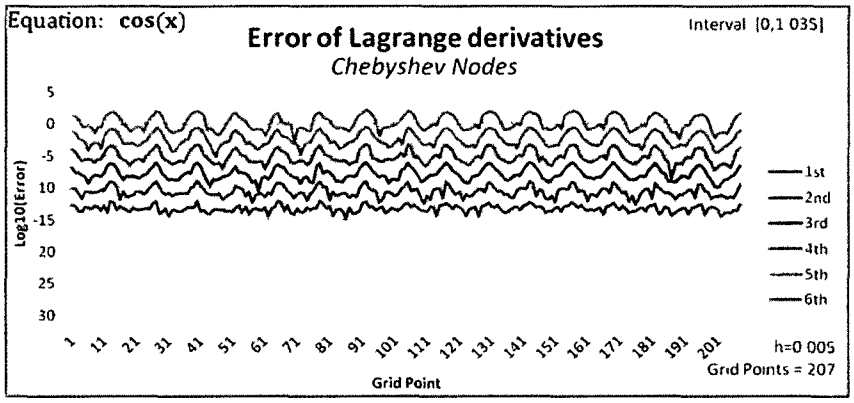


(c) Sixth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

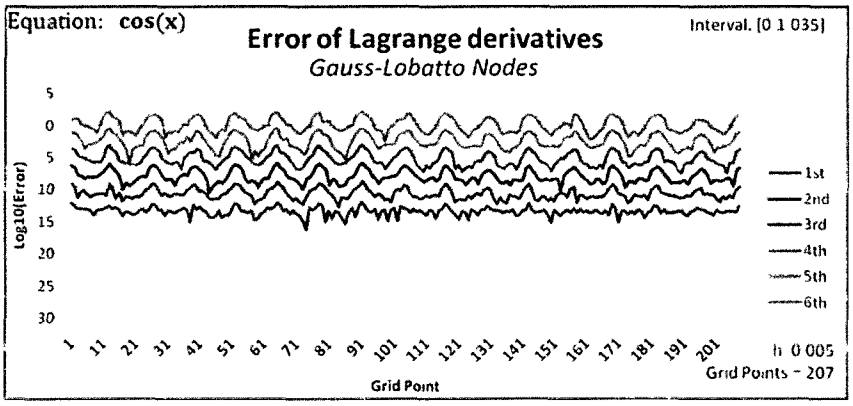
Figure B.6: Central difference approximations of the Laplace operator applied to the function $f(x) = \cos(x)$, over the interval $[0, 10.35]$, utilizing 208 total grid points, and various orders of accuracy.



(a) Differentiated Lagrange interpolating polynomials using equally spaced nodes to compute the first- through sixth-order derivatives.

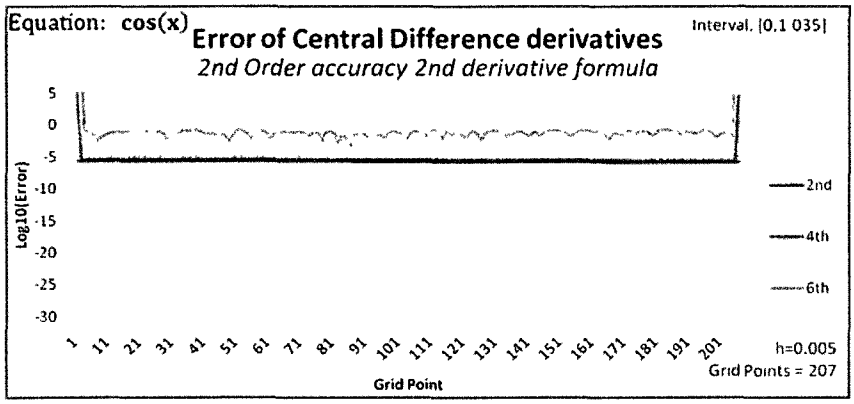


(b) Differentiated Lagrange interpolating polynomials using the Chebyshev nodes to compute the first- through sixth-order derivatives.

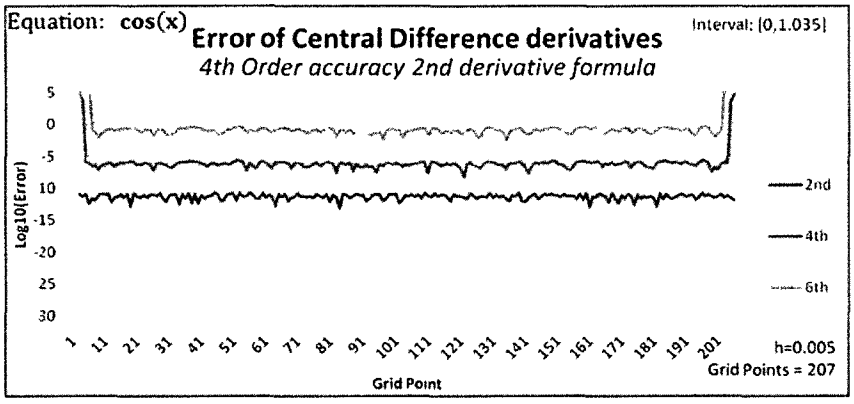


(c) Differentiated Lagrange interpolating polynomials using the Gauss-Lobatto nodes to compute the first- through sixth-order derivatives.

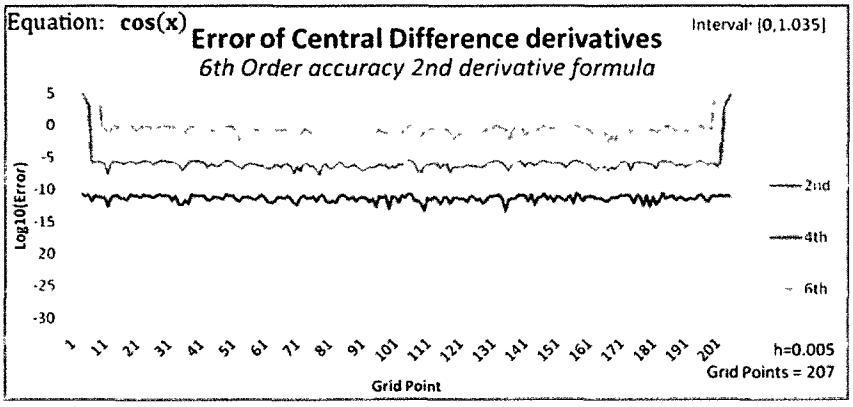
Figure B.7: Differentiated piecewise twelfth degree Lagrange interpolating polynomials for the function $f(x) = \cos(x)$, over the interval $[0, 1.035]$, utilizing 208 total grid points, and three different grid spacings.



(a) Second-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.



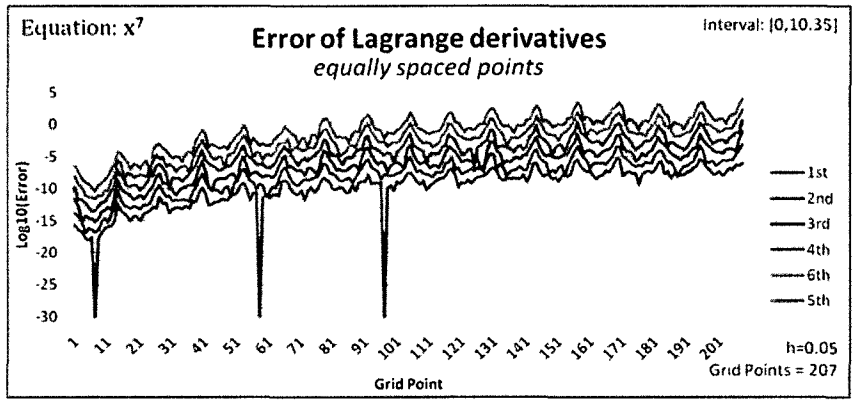
(b) Fourth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.



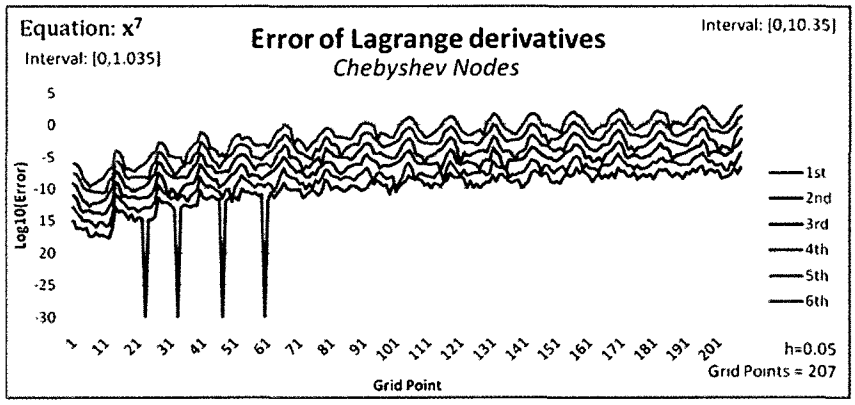
(c) Sixth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

Figure B.8: Central difference approximations of the Laplace operator applied to the function $f(x) = \cos(x)$, over the interval $[0, 1.035]$, utilizing 208 total grid points, and various orders of accuracy.

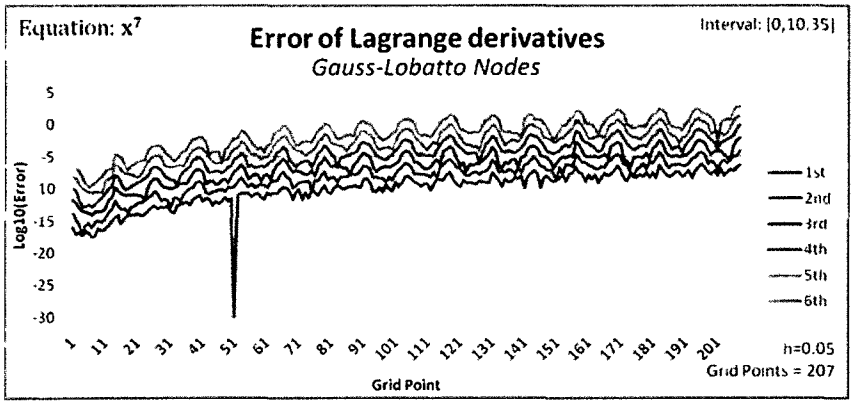
B.3 Polynomial Function



(a) Differentiated Lagrange interpolating polynomials using equally spaced nodes to compute the first- through sixth-order derivatives.

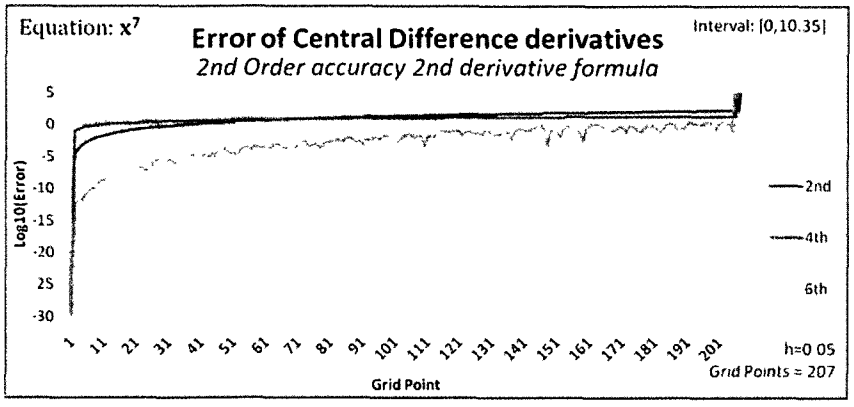


(b) Differentiated Lagrange interpolating polynomials using the Chebyshev nodes to compute the first- through sixth-order derivatives.

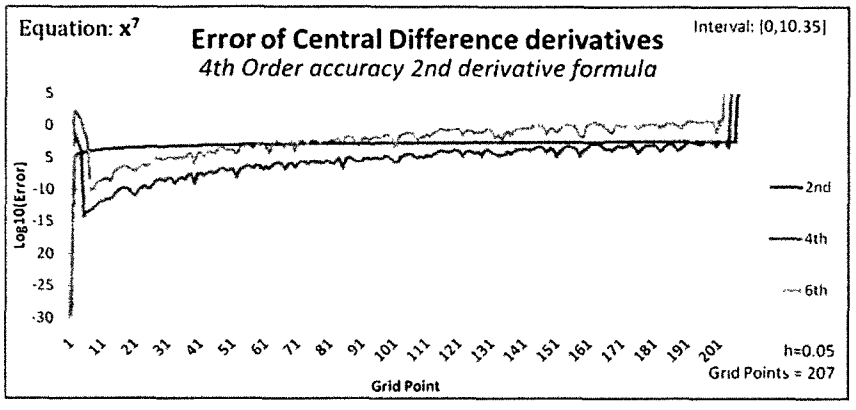


(c) Differentiated Lagrange interpolating polynomials using the Gauss-Lobatto nodes to compute the first- through sixth-order derivatives.

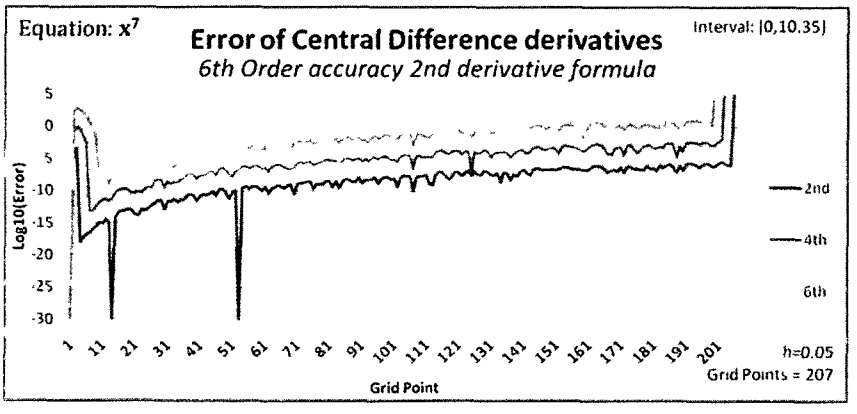
Figure B.9: Differentiated piecewise twelfth degree Lagrange interpolating polynomials for the function $f(x) = x^7$, over the interval $[0, 10.35]$, utilizing 208 total grid points, and three different grid spacings.



(a) Second-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

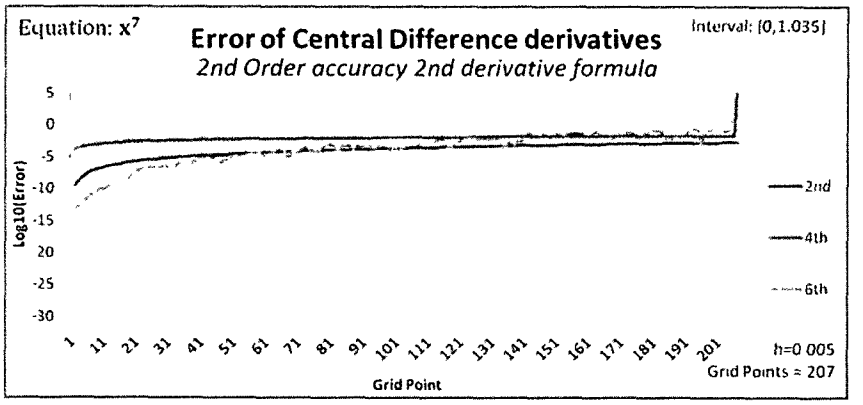


(b) Fourth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

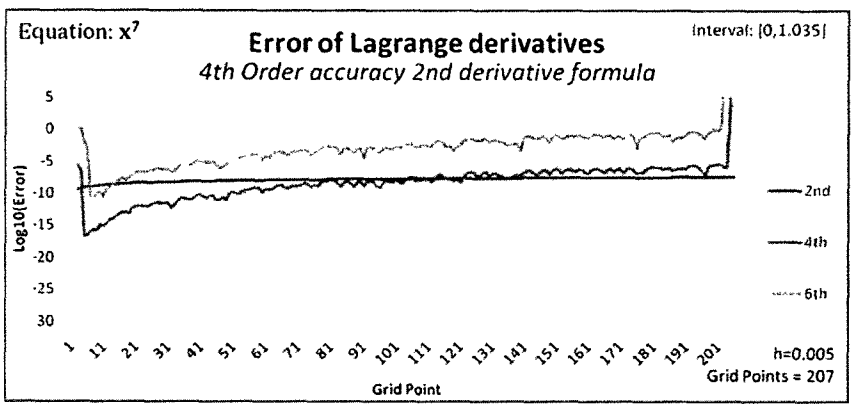


(c) Sixth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

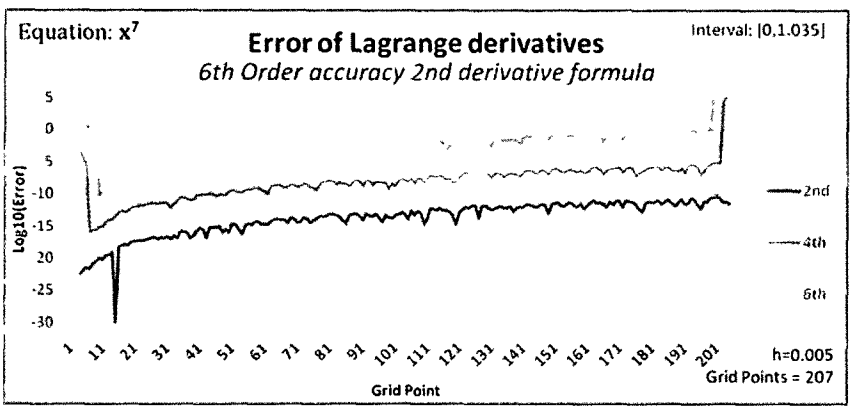
Figure B.10: Central difference approximations of the Laplace operator applied to the function $f(x) = x^7$, over the interval $[0, 10.35]$, utilizing 208 total grid points, and various orders of accuracy.



(a) Second-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

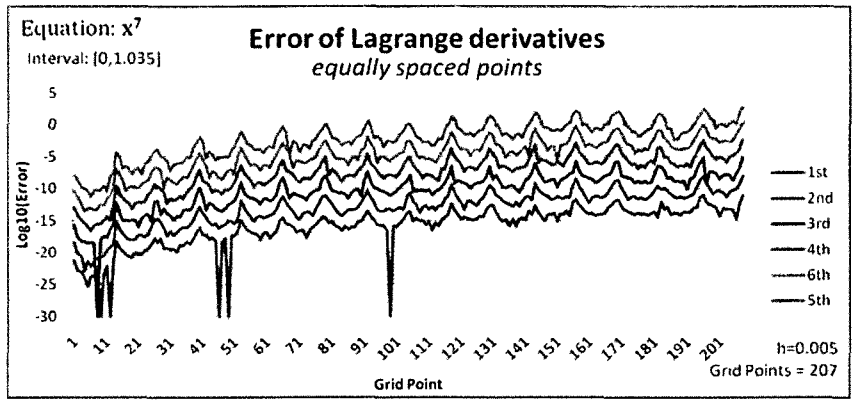


(b) Fourth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

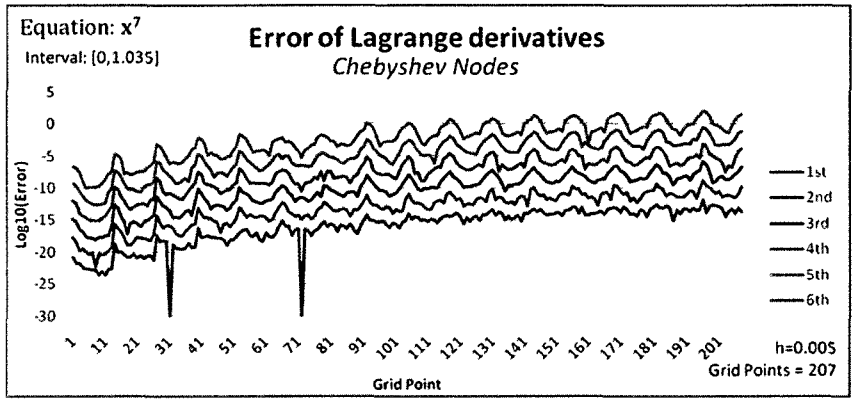


(c) Sixth-order accurate central difference approximation of the Laplace operator used to compute the second-, fourth-, and sixth-order derivatives.

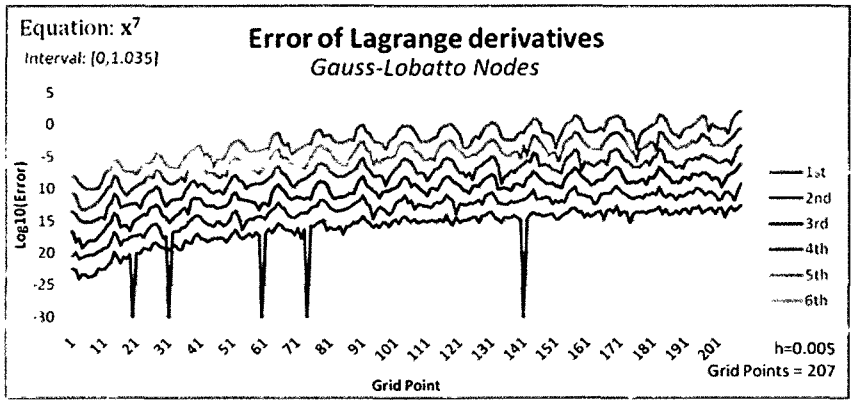
Figure B.11: Central difference approximations of the Laplace operator applied to the function $f(x) = x^7$, over the interval $[0, 1.035]$, utilizing 208 total grid points, and various orders of accuracy.



(a) Differentiated Lagrange interpolating polynomials using equally spaced nodes to compute the first- through sixth-order derivatives.



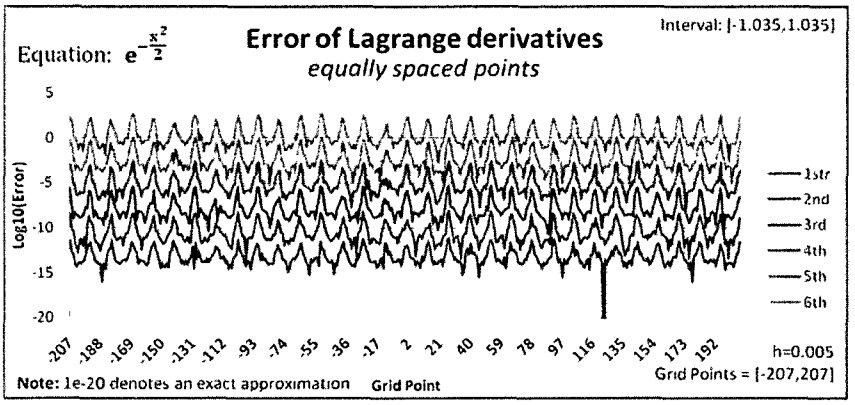
(b) Differentiated Lagrange interpolating polynomials using the Chebyshev nodes to compute the first- through sixth-order derivatives.



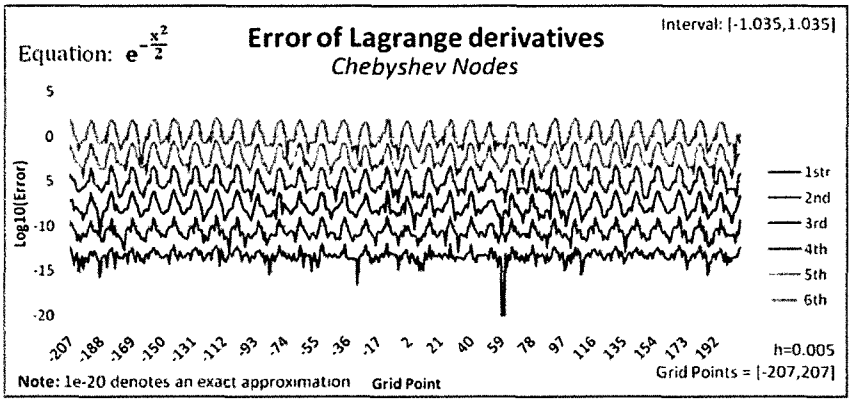
(c) Differentiated Lagrange interpolating polynomials using the Gauss-Lobatto nodes to compute the first- through sixth-order derivatives.

Figure B.12: Differentiated piecewise twelfth degree Lagrange interpolating polynomials for the function $f(x) = x^7$, over the interval $[0, 1.035]$, utilizing 208 total grid points, and three different grid spacings.

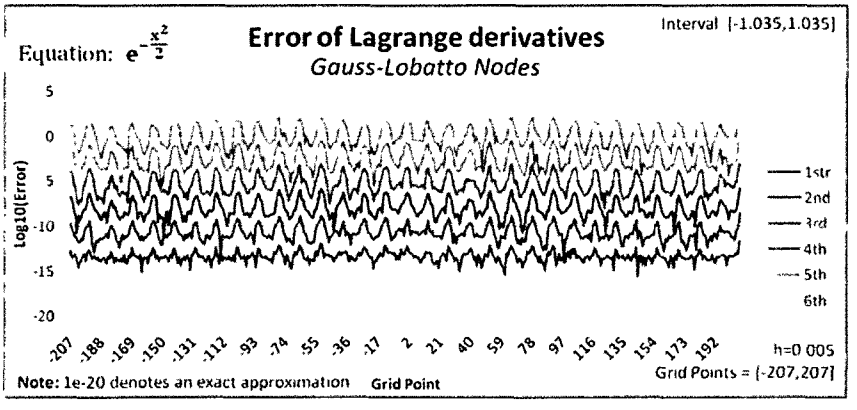
B.4 Gaussian Function over a Symmetric Interval



(a) Differentiated Lagrange interpolating polynomials using equally spaced nodes to compute the first- through sixth-order derivatives.

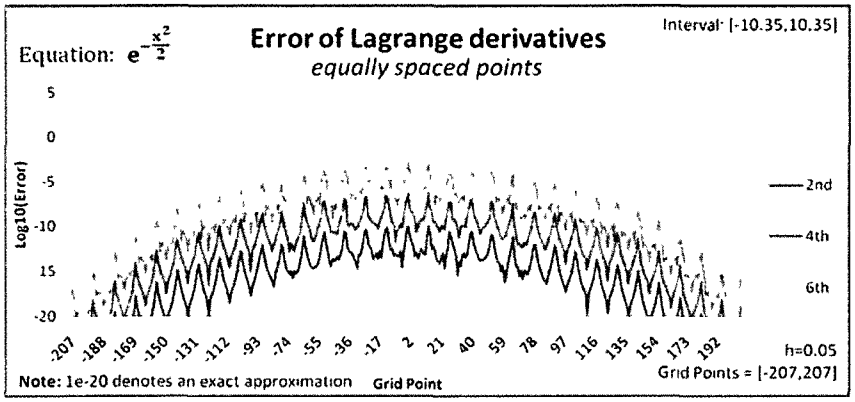


(b) Differentiated Lagrange interpolating polynomials using the Chebyshev nodes to compute the first- through sixth-order derivatives.

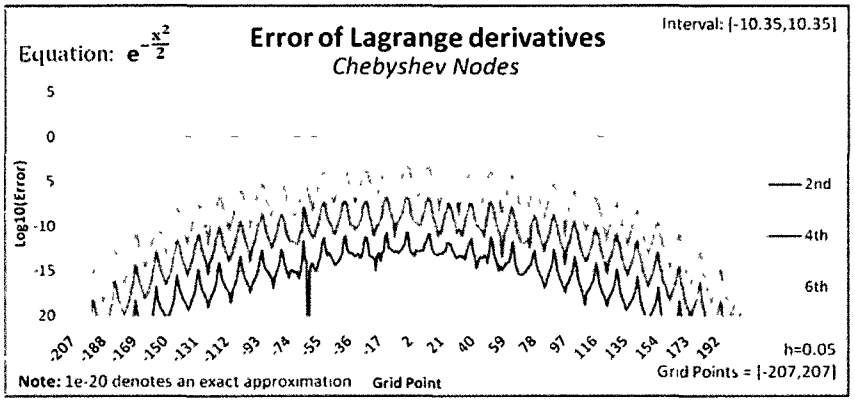


(c) Differentiated Lagrange interpolating polynomials using the Gauss-Lobatto nodes to compute the first- through sixth-order derivatives.

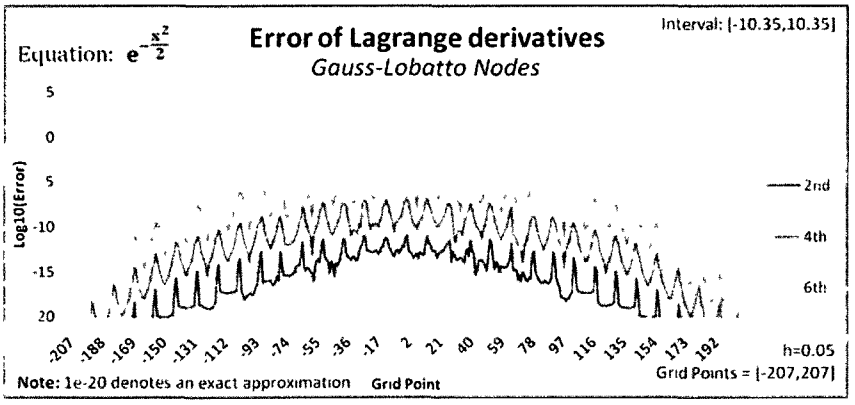
Figure B.13: Differentiated piecewise twelfth degree Lagrange interpolating polynomials for the function $f(x) = e^{(-\frac{x^2}{2})}$, over the interval $[-1.035, 1.035]$, utilizing 415 total grid points.



(a) Differentiated Lagrange interpolating polynomials using equally spaced nodes to compute the first- through sixth-order derivatives.



(b) Differentiated Lagrange interpolating polynomials using the Chebyshev nodes to compute the first- through sixth-order derivatives.



(c) Differentiated Lagrange interpolating polynomials using the Gauss-Lobatto nodes to compute the first- through sixth-order derivatives.

Figure B.14: Differentiated piecewise twelfth degree Lagrange interpolating polynomials for the function $f(x) = e^{(-\frac{x^2}{2})}$, over the interval $[-10.35, 10.35]$, utilizing 415 total grid points.

APPENDIX C

FORTRAN SOURCE CODE

C.1 Model Problem Using Second-Order Accurate Scheme

```
1 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
2 c Model problem source code
3 c
4 c This code utilizes a second-order accurate central difference
5 c approximation of the Laplace operator. By default the program
6 c utilizes the parameter N=3 for the Generalized FDTD Scheme.
7 c One must comment out the appropriate sections of code should
8 c a lower N value be desired.
9 c
10 c This code was adapated from code written by
11 c       Weizhong Dai and Fred Moxley (c) 2011
12 c This code was modified by James Elliott (c) 2011
13 c
14 c This is the beginning of the main program
15 c all variables must be declared before any assignments are made
16 c parameter values may only be assigned here, and may never be
     changed (they are constants)
17 c
18 c ********************* IMPORTANT *********************
19 c ALWAYS write real values using ### ###D##
20 c "D" ensures the values will be double precision
21 c alternatively, when compiling the program, utilize the flags
22 c gfortran -O -fdefault-real-8 -fdefault-double-8 -frange-check -
     Wall
23 c this will ensure all real values are double precision,
24 c as well as enable useful warnings such as unused variables or
25 c loops running past an array's limits
26 c ********************* IMPORTANT *********************
27
28       implicit none
29 c Do not specify KE, instead change ddx,
30 c KE will be computed to be over the interval [0,1]
31 c Input  ddx - the spatial step for the x direction
32 c Input: ra  - the mesh ratio, see Table 5.1 for suitable mesh
     ratios
33       integer KE
34       double precision ddx, ra
35       parameter(ddx=1.0D-2, KE=1.0D0/ddx, ra=0.25D0)
36 c Declare the size of the arrays
37 c u_r is an array for the real component of the wavefunction
38 c u_i is an array for the imaginary component of the wavefunction
39 c u_r#p corresponds to an #th derivative of the real component
40 c u_i#p corresponds to an #th derivative of the imaginary
     component
41       dimension u_r(0:KE),u_i(0:KE),vp(0:KE),
42      &        u_r2p(0:KE),u_r4p(0:KE),u_r6p(0:KE),
43      &        u_r8p(0:KE),u_r10p(0:KE),u_r12p(0.KE),
44      &        u_r14p(0:KE),u_i2p(0:KE),u_i4p(0:KE),
45      &        u_i6p(0:KE),u_i8p(0:KE),u_i10p(0:KE),
46      &        u_i12p(0:KE),u_i14p(0:KE),x(0.KE),
47 c Ensure this array is large enough to to hold 1/dt values
48 c (or reprogram the method to only store the largest values)
```

```
49      &       error_max(0:1000000)
50 c Declare the type for each variable
51        double precision e_r, e_i, err_r, err_i, error_max, err_max
              ,
52      & u_r, u_r2p, u_r4p, u_r6p, u_r8p, u_r10p, u_r12p, u_r14p,
53      & u_i, u_i2p, u_i4p, u_i6p, u_i8p, u_i10p, u_i12p, u_i14p,
54      & vp,pi,pi2,dt,cc,
55      & cvh1,cvh2,cvh3,cvh4,cvh5,cvh6,cvh7,
56      & r1,r2,r3,r4,r5,r6,r7,
57      & ddx2,x,
58      & tnh,tn1,tnh1,
59      & emax_r, emax_i
60 c These values are the coefficients of the taylor series
61        double precision c1_24,c3,c5,c10,c21,c35,c7,c1_322560,
              c1_1920
62
63 c k_start, and k_end represent the lower and upper limits of
64 c the computable grid points e.g., 1 to KE-1
65 c skipN is used to restrict the number of lines outputed
66        integer k_start,k_end,skipN
67 c spatial counter k, and temporal counter n
68 c n_max is the timestep with the largest error
69 c nsteps is the total number of timesteps
70        integer k,n_max,nsteps,n
71
72 c Variable initializations, no more variables may be declared.
73        pi=3.14159265358979323846D0
74        pi2=pi*pi
75 c Compute dt based on ddx and the mesh ratio
76        dt=ddx*ddx*ra
77        ddx2=ddx*ddx
78        r1=dt/(ddx*ddx)
79        r2=r1*r1
80        r3=r2*r1
81        r4=r3*r1
82        r5=r4*r1
83        r6=r5*r1
84        r7=r6*r1
85        cc=1.0D0
86 c this enforces that 0 < t <= 1
87        nsteps=int(1.0D0/dt)
88 c configure how many lines are outputed total
89 c if n mod skipN == 0, then output
90        skipN = ceiling(nsteps/32000 0)
91
92 c Coefficients in the taylor expansion, to ensure double
     precision
93        c1_24=1.0D0/24.0D0
94        c3=3.0D0
95        c5=5.0D0
96        c10=10.0D0
97        c21=21.0D0
98        c35=35.0D0
```

```
 99       c7=7.0D0
100       c1_322560=1.0D0/322560.0D0
101       c1_1920=1.0D0/1920.0D0
102
103       print *,"KE: ",KE, ", dx: ", ddx, ", ORDER ",2,
104      &        ", nsteps  ', nsteps
105 c configure the computable points
106       k_start=1
107       k_end=KE-1
108
109 c Apply the initial conditions
110       do k=0,KE
111         u_r(k)=0.0D0
112         u_i(k)=0.0D0
113         vp(k)=0.0D0
114       enddo
115
116       do k=0,KE
117         vp(k)=pi2
118       enddo
119 c discretize the spatial domain in KE+1 intervals of size dx
120       do k=0,KE
121         x(k)=k*ddx
122       enddo
123 c Apply the initial conditions for the wavefunction
124       do k=0,KE
125         u_r(k)=sin(pi*x(k))
126         u_i(k)=-sin(dt*pi2)*sin(pi*x(k))
127       enddo
128
129 c This output matches the output in the error calculation
130 c this effectively makes the program output a csv file should one
131 c run the program from the commandline and
132 c capture the output in a file
133       print *, "Timestep, time (s). MaxError"
134 c start time level
135       do n=1,nsteps
136       tnh=2.0D0*(n-0.5D0)*dt*pi2
137
138 c calculate the derivatives of imaginary values
139       call calsed(u_i,u_i2p,KE)
140       call calsed(u_i2p,u_i4p,KE)
141       call calsed(u_i4p,u_i6p,KE)
142       call calsed(u_i6p,u_i8p,KE)
143       call calsed(u_i8p,u_i10p,KE)
144       call calsed(u_i10p,u_i12p,KE)
145       call calsed(u_i12p,u_i14p,KE)
146
147 c begin calculating the realpart
148       do k=k_start,k_end
149       cvh1=vp(k)*dt
150       cvh2=cvh1*cvh1
151       cvh3=cvh2*cvh1
```

```
152       cvh4=cvh3*cvh1
153       cvh5=cvh4*cvh1
154       cvh6=cvh5*cvh1
155       cvh7=cvh6*cvh1
156
157 c calculate the realvalue
158       u_r(k)=u_r(k)-r1*u_i2p(k)+cvh1*u_i(k)
159 c p=1
160      $    +c1_24*(          r3*u_i6p(k)
161      &            -c3*cvh1*r2*u_i4p(k)
162      &            +c3*cvh2*r1*u_i2p(k)
163      &               -cvh3*u_i(k))
164 c p=2
165      &  -c1_1920*(            r5*u_i10p(k)
166      &               -c5*cvh1*r4*u_i8p(k)
167      &              +c10*cvh2*r3*u_i6p(k)
168      &              -c10*cvh3*r2*u_i4p(k)
169      &              + c5*cvh4*r1*u_i2p(k)
170      &                  -cvh5*u_i(k))
171 c p=3
172      &  +cc*c1_322560*(           r7*u_i14p(k)
173      &                   -c7*cvh1*r6*u_i12p(k)
174      &                  +c21*cvh2*r5*u_i10p(k)
175      &                  -c35*cvh3*r4*u_i8p(k)
176      &                  +c35*cvh4*r3*u_i6p(k)
177      &                  -c21*cvh5*r2*u_i4p(k)
178      &                   +c7*cvh6*r1*u_i2p(k)
179      &                      -cvh7*u_i(k))
180       enddo
181
182 c calculate the derivatives of realvalue
183
184       call calsed(u_r,u_r2p,KE)
185       call calsed(u_r2p,u_r4p,KE)
186       call calsed(u_r4p,u_r6p,KE)
187       call calsed(u_r6p,u_r8p,KE)
188       call calsed(u_r8p,u_r10p,KE)
189       call calsed(u_r10p,u_r12p,KE)
190       call calsed(u_r12p,u_r14p,KE)
191
192 c begin calculating the imaginary part
193       do k=k_start,k_end
194       cvh1=vp(k)*dt
195       cvh2=cvh1*cvh1
196       cvh3=cvh2*cvh1
197       cvh4=cvh3*cvh1
198       cvh5=cvh4*cvh1
199       cvh6=cvh5*cvh1
200       cvh7=cvh6*cvh1
201
202 c calculate the imaginary values
203         u_i(k)=u_i(k)+u_r2p(k)*r1-cvh1*u_r(k)
204 c p=1
```

```
205       $  -c1_24*(          r3*u_r6p(k)
206       &           -c3*cvh1*r2*u_r4p(k)
207       &           +c3*cvh2*r1*u_r2p(k)
208       &              -cvh3*u_r(k))
209 c p=2
210       &  +c1_1920*(          r5*u_r10p(k)
211       &             -c5*cvh1*r4*u_r8p(k)
212       &            +c10*cvh2*r3*u_r6p(k)
213       &            -c10*cvh3*r2*u_r4p(k)
214       &             +c5*cvh4*r1*u_r2p(k)
215       &             -cvh5*u_r(k))
216 c p=3
217       &  -cc*c1_322560*(          r7*u_r14p(k)
218       &                  -c7*cvh1*r6*u_r12p(k)
219       &                 +c21*cvh2*r5*u_r10p(k)
220       &                 -c35*cvh3*r4*u_r8p(k)
221       &                 +c35*cvh4*r3*u_r6p(k)
222       &                 -c21*cvh5*r2*u_r4p(k)
223       &                  +c7*cvh6*r1*u_r2p(k)
224       &                     -cvh7*u_r(k))
225        enddo
226
227 c Exact solution components
228        tn1=2.0D0*n*dt*pi2
229        tnh1=2.0D0*(n+0.5D0)*dt*pi2
230
231 c calculate the exact solution, and determine
232 c the max error for this iteration
233        emax_r=0.0D0
234        emax_i=0.0D0
235        do k=k_start,k_end
236          e_r = dcos(tn1)*dsin(pi*x(k))
237          e_i = -dsin(tnh1)*dsin(pi*x(k))
238
239          err_r = dabs(u_r(k)-e_r)
240          err_i = dabs(u_i(k)-e_i)
241
242          if(emax_r .le. err_r) then
243            emax_r = err_r
244          endif
245
246          if(emax_i .le. err_i) then
247            emax_i = err_i
248          endif
249        enddo
250 c determine the max error, this could be rewritten using the max
     ()
251 c intrinsic function
252        if(emax_r .le. emax_i) then
253          error_max(n) = emax_i
254        else
255          error_max(n) = emax_r
256        endif
```

```
257 c print the largest error obtained within this timestep
258 c the conditionals may be removed, the logic ensures
259 c that no more than 32000 lines are ouputed, which makes plotting
260 c and file size much smaller when working with very small mesh
261 c ratios  skipN may be changed at the start of the program if
262 c one wishes more or fewer lines of output
263       if(nsteps  lt  32000) then
264         print *, n, ,', n*dt,', , error_max(n)
265       else
266         if(mod(n,skipN)  eq  0) ther
267           print *, n,'  , n*dt,',', error_max(n)
268         endif
269       endif
270
271 c end of the time loop
272       enddo
273
274 c determine the largest error observed in the entire simulation
275       err_max=0 0D0
276       do n=1,nsteps
277         if(err_max LE error_max(n)) then
278           err_max=error_max(n)
279           n_max=n
280         endif
281       enddo
282       print *,  Large t Error  ', n_max, err_max
283 c write all errors, and assocaited time intervals to a file
284 c this may be commented out, if one is running the program from
285 c the command line and piping the output in a file
286       open (unit=22,file='error_max_FD1D_N_3_0_2_x_200_ra_1 00
             dat )
287       do  n=1,nsteps
288         write(22,10)n*dt, error_max(n)
289       enddo
290       close(22)
291   10  format(f10 8,1x,F20 16)
292
293 c This is the end of the main program
294       stop
295       end
296
297 c This routine computes the 2nd derivaitve
298 c input  f - the function to differentiate
299 c input  KE - the number of grid points, indexed from zero
300 c output  f" in the array fdp
301       subroutine calsed(f,fdp,KE)
302        mplic  none
303       dimension f(0 KE),fdp(0 KE)
304       double pre  ion f,fdp
305       integer k,ke
306
307 c Compute a second-order accurate central difference
308       d  k=1,KE-1
```

```
309           fdp(k)=f(k-1)-2.0D0*f(k)+f(k+1)
310         enddo
311 c This is the end of the subroutine calsed
312         return
313         end
```

Listing C.1: Model problem using the second-order accurate scheme

C.2 Model Problem Using Sixth-Order Accurate Scheme

```
1 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
2 c Model problem source code
3 c
4 c This code utilizes a sixth-order accurate central difference
5 c approximation of the Laplace operator. By default the program
6 c utilizes the parameter N=3 for the Generalized FDTD Scheme.
7 c One must comment out the appropriate sections of code should
8 c a lower N value be desired.
9 c
10 c This code was adapated from code written by
11 c       Weizhong Dai and Fred Moxley (c) 2011
12 c
13 c This code was modified by James Elliott (c) 2011
14 c
15 c This is the beginning of the main program
16 c all variables must be declared before any assignments are made
17 c parameter values may only be assigned here, and may never be
18 c changed (they are constants)
19 c
20 c ********************* IMPORTANT *********************
21 c ALWAYS write real values using ###.###D##
22 c "D" ensures the values will be double precision
23 c alternatively, when compiling the program, utilize the flags
24 c gfortran -O -fdefault-real-8 -fdefault-double-8 -frange-check -
     Wall
25 c this will ensure all real values are double precision,
26 c as well as enable useful warnings such as unused variables or
27 c loops running past an array's limits
28 c ********************* IMPORTANT *********************
29
30       implicit none
31 c Do not specify KE, instead change ddx,
32 c KE will be computed to be over the interval [0,1]
33 c Input: ddx - the spatial step for the x direction
34 c Input. ra  - the mesh ratio, see Table 5 1 for suitable mesh
     ratios
35       integer KE
36       double precision ddx, ra
37       parameter(ddx=1.0D-2, KE=1.0D0/ddx, ra=0.25D0)
38 c Declare the size of the arrays
39 c u_r is an array for the real component of the wavefunction
40 c u_i is an array for the imaginary component of the wavefunction
41 c u_r#p corresponds to an #th derivative of the real component
42 c u_i#p corresponds to an #th derivative of the imaginary
     component
43       dimension vp(0:KE), x(0:KE),
44      & u_r(0:KE),  u_r2p(0:KE), u_r4p(0:KE), u_r6p(0:KE),
45      & u_r8p(0:KE),u_r10p(0:KE),u_r12p(0:KE),u_r14p(0:KE),
46      & u_i(0:KE),  u_i2p(0:KE), u_i4p(0:KE), u_i6p(0:KE),
47      & u_i8p(0:KE),u_i10p(0:KE),u_i12p(0:KE),u_i14p(0:KE),
48 c Ensure this array is large enough to to hold 1/dt values
49 c (or reprogram the method to only store the largest values)
```

```
50      &       error_max(0:1000000)
51 c Declare the type for each variable
52        double precision e_r, e_i, err_r, err_i, error_max, err_max
             ,
53      & u_r, u_r2p, u_r4p, u_r6p, u_r8p, u_r10p, u_r12p, u_r14p,
54      & u_i, u_i2p, u_i4p, u_i6p, u_i8p, u_i10p, u_i12p, u_i14p,
55      & vp,pi,pi2,dt,cc,
56      & cvh1,cvh2,cvh3,cvh4,cvh5,cvh6,cvh7,
57      & r1,r2,r3,r4,r5,r6,r7,
58      & ddx2,x,
59      & tnh,tn1,tnh1,
60      & emax_r, emax_i
61 c These values are the coefficients of the taylor series
62        double precision c1_24,c3,c5,c10,c21,c35,c7,c1_322560,
             c1_1920
63
64 c k_start, and k_end represent the lower and upper limits of
65 c the computable grid points e g., 3 to KE-3
66 c skipN is used to restrict the number of lines outputed
67        integer k_start,k_end,skipN
68 c spatial counter k, and temporal counter n
69 c n_max is the timestep with the largest error
70 c nsteps is the total number of timesteps
71        integer k,n_max,nsteps,n
72
73 c Variable initializations, no more variables may be declared.
74        pi=3.14159265358979323846D0
75        pi2=pi*pi
76 c Compute dt based on ddx and the mesh ratio
77        dt=ddx*ddx*ra
78        ddx2=ddx*ddx
79        r1=dt/(ddx*ddx)
80        r2=r1*r1
81        r3=r2*r1
82        r4=r3*r1
83        r5=r4*r1
84        r6=r5*r1
85        r7=r6*r1
86        cc=1.0D0
87 c this enforces that 0 < t <= 1
88        nsteps=int(1.0D0/dt)
89 c configure how many lines are outputed total
90 c if n mod skipN == 0, then output
91        skipN = ceiling(nsteps/32000.0)
92
93 c Coefficients in the taylor expansion, to ensure double
     precision
94        c1_24=1.0D0/24.0D0
95        c3=3.0D0
96        c5=5 0D0
97        c10=10.0D0
98        c21=21 0D0
99        c35=35.0D0
```

```
100       c7=7.0D0
101       c1_322560=1.0D0/322560.0D0
102       c1_1920=1.0D0/1920.0D0
103
104       print *,"KE: ",KE, ", dx: ", ddx, ", ORDER ",6,
105      &        ", nsteps  ", nsteps
106 c configure the computable points
107       k_start=3
108       k_end=KE-3
109
110 c Apply the initial conditions
111       do k=0,KE
112         u_r(k)=0.0D0
113         u_i(k)=0.0D0
114         vp(k)=0.0D0
115       enddo
116
117       do k=0,KE
118         vp(k)=pi2
119       enddo
120 c discretize the spatial domain in KE+1 intervals of size dx
121       do k=0,KE
122         x(k)=k*ddx
123       enddo
124 c Apply the initial conditions for the wavefunction
125
126
127       do k=0,KE
128         u_r(k)=sin(pi*x(k))
129         u_i(k)=-sin(dt*pi2)*sin(pi*x(k))
130       enddo
131
132 c This output matches the output in the error calculation
133 c this effectively makes the program output a csv file should one
134 c run the program from the commandline and
135 c capture the output in a file
136       print *, "Timestep, time (s), MaxError"
137 c start time level
138
139       do n=1,nsteps
140       tnh=2.0D0*(n-0.5D0)*dt*pi2
141
142 c calculate the derivatives of imaginary values
143 c and compute exact values for the uncomputable points
144       call calsed(u_i,u_i2p,KE)
145         u_i2p(1)=ddx2*pi2*sin(tnh)*sin(pi*x(1))
146         u_i2p(2)=ddx2*pi2*sin(tnh)*sin(pi*x(2))
147         u_i2p(KE-1)=ddx2*pi2*sin(tnh)*sin(pi*x(KE-1))
148         u_i2p(KE-2)=ddx2*pi2*sin(tnh)*sin(pi*x(KE-2))
149
150       call calsed(u_i2p,u_i4p,KE)
151         u_i4p(1)=-ddx2*pi2*u_i2p(1)
152         u_i4p(2)=-ddx2*pi2*u_i2p(2)
```

```
153         u_14p(KE-1)=-ddx2*p12*u_12p(KE-1)
154         u_14p(KE-2)=-ddx2*p12*u_12p(KE-2)
155
156       call calsed(u_14p,u_16p,KE)
157         u_16p(1)=-ddx2*p12*u_14p(1)
158         u_16p(2)=-ddx2*p12*u_14p(2)
159         u_16p(KE-1)=-ddx2*p12*u_14p(KE-1)
160         u_16p(KE-2)=-ddx2*p12*u_14p(KE-2)
161
162       call calsed(u_16p,u_18p,KE)
163         u_18p(1)=-ddx2*p12*u_16p(1)
164         u_18p(2)=-ddx2*p12*u_16p(2)
165         u_18p(KE-1)=-ddx2*p12*u_16p(KE-1)
166         u_18p(KE-2)=-ddx2*p12*u_16p(KE-2)
167
168       call calsed(u_18p,u_110p,KE)
169         u_110p(1)=-ddx2*p12*u_18p(1)
170         u_110p(2)=-ddx2*p12*u_18p(2)
171         u_110p(KE-1)=-ddx2*p12*u_18p(KE-1)
172         u_110p(KE-2)=-ddx2*p12*u_18p(KE-2)
173
174       call calsed(u_110p,u_112p,KE)
175         u_112p(1)=-ddx2*p12*u_110p(1)
176         u_112p(2)=-ddx2*p12*u_110p(2)
177         u_112p(KE-1)=-ddx2*p12*u_110p(KE-1)
178         u_112p(KE-2)=-ddx2*p12*u_110p(KE-2)
179
180       call calsed(u_112p,u_114p,KE)
181         u_114p(1)=-ddx2*p12*u_112p(1)
182         u_114p(2)=-ddx2*p12*u_112p(2)
183         u_114p(KE-1)=-ddx2*p12*u_112p(KE-1)
184         u_114p(KE-2)=-ddx2*p12*u_112p(KE-2)
185
186 c begin calculating the realpart
187       do k=k_start,k_end
188       cvh1=vp(k)*dt
189       cvh2=cvh1*cvh1
190       cvh3=cvh2*cvh1
191       cvh4=cvh3*cvh1
192       cvh5=cvh4*cvh1
193       cvh6=cvh5*cvh1
194       cvh7=cvh6*cvh1
195
196 c calculate the realvalue
197       u_r(k)=u_r(k)-r1*u_12p(k)+cvh1*u_1(k)
198 c p=1
199      $   +c1_24*(          r3*u_16p(k)
200      &           -c3*cvh1*r2*u_14p(k)
201      &           +c3*cvh2*r1*u_12p(k)
202      &              -cvh3*u_1(k))
203 c p=2
204      &  -c1_1920*(            r5*u_110p(k)
205      &              -c5*cvh1*r4*u_18p(k)
```

```
206      &              +c10*cvh2*r3*u_i6p(k)
207      &              -c10*cvh3*r2*u_i4p(k)
208      &              + c5*cvh4*r1*u_i2p(k)
209      &                  -cvh5*u_i(k))
210 c p=3
211      &  +cc*c1_322560*(          r7*u_i14p(k)
212      &                   -c7*cvh1*r6*u_i12p(k)
213      &                  +c21*cvh2*r5*u_i10p(k)
214      &                  -c35*cvh3*r4*u_i8p(k)
215      &                  +c35*cvh4*r3*u_i6p(k)
216      &                  -c21*cvh5*r2*u_i4p(k)
217      &                   +c7*cvh6*r1*u_i2p(k)
218      &                      -cvh7*u_i(k))
219       enddo
220 c provide exact values for th uncomputable points
221       tn1=2.0D0*n*dt*pi2
222         u_r(1)=cos(tn1)*sin(pi*x(1))
223         u_r(2)=cos(tn1)*sin(pi*x(2))
224         u_r(KE-1)=cos(tn1)*sin(pi*x(KE-1))
225         u_r(KE-2)=cos(tn1)*sin(pi*x(KE-2))
226
227 c calculate the derivatives of real values
228
229 c and provide exact values for the uncomputable points
230       call calsed(u_r,u_r2p,KE)
231         u_r2p(1)=-ddx2*pi2*u_r(1)
232         u_r2p(2)=-ddx2*pi2*u_r(2)
233         u_r2p(KE-1)=-ddx2*pi2*u_r(KE-1)
234         u_r2p(KE-2)=-ddx2*pi2*u_r(KE-2)
235
236       call calsed(u_r2p,u_r4p,KE)
237         u_r4p(1)=-ddx2*pi2*u_r2p(1)
238         u_r4p(2)=-ddx2*pi2*u_r2p(2)
239         u_r4p(KE-1)=-ddx2*pi2*u_r2p(KE-1)
240         u_r4p(KE-2)=-ddx2*pi2*u_r2p(KE-2)
241
242       call calsed(u_r4p,u_r6p,KE)
243         u_r6p(1)=-ddx2*pi2*u_r4p(1)
244         u_r6p(2)=-ddx2*pi2*u_r4p(2)
245         u_r6p(KE-1)=-ddx2*pi2*u_r4p(KE-1)
246         u_r6p(KE-2)=-ddx2*pi2*u_r4p(KE-2)
247
248       call calsed(u_r6p,u_r8p,KE)
249         u_r8p(1)=-ddx2*pi2*u_r6p(1)
250         u_r8p(2)=-ddx2*pi2*u_r6p(2)
251         u_r8p(KE-1)=-ddx2*pi2*u_r6p(KE-1)
252         u_r8p(KE-2)=-ddx2*pi2*u_r6p(KE-2)
253
254       call calsed(u_r8p,u_r10p,KE)
255         u_r10p(1)=-ddx2*pi2*u_r8p(1)
256         u_r10p(2)=-ddx2*pi2*u_r8p(2)
257         u_r10p(KE-1)=-ddx2*pi2*u_r8p(KE-1)
258         u_r10p(KE-2)=-ddx2*pi2*u_r8p(KE-2)
```

```
259
260       call calsed(u_r10p,u_r12p,KE)
261         u_r12p(1)=-ddx2*pi2*u_r10p(1)
262         u_r12p(2)=-ddx2*pi2*u_r10p(2)
263         u_r12p(KE-1)=-ddx2*pi2*u_r10p(KE-1)
264         u_r12p(KE-2)=-ddx2*pi2*u_r10p(KE-2)
265
266       call calsed(u_r12p,u_r14p,KE)
267         u_r14p(1)=-ddx2*pi2*u_r12p(1)
268         u_r14p(2)=-ddx2*pi2*u_r12p(2)
269         u_r14p(KE-1)=-ddx2*pi2*u_r12p(KE-1)
270         u_r14p(KE-2)=-ddx2*pi2*u_r12p(KE-2)
271
272 c begin calculating the imaginary part
273       do k=k_start,k_end
274       cvh1=vp(k)*dt
275       cvh2=cvh1*cvh1
276       cvh3=cvh2*cvh1
277       cvh4=cvh3*cvh1
278       cvh5=cvh4*cvh1
279       cvh6=cvh5*cvh1
280       cvh7=cvh6*cvh1
281
282 c calculate the imaginary values
283         u_i(k)=u_i(k)+u_r2p(k)*r1-cvh1*u_r(k)
284 c p=1
285      $  -c1_24*(          r3*u_r6p(k)
286      &          -c3*cvh1*r2*u_r4p(k)
287      &          +c3*cvh2*r1*u_r2p(k)
288      &             -cvh3*u_r(k))
289 c p=2
290      &  +c1_1920*(          r5*u_r10p(k)
291      &            -c5*cvh1*r4*u_r8p(k)
292      &           +c10*cvh2*r3*u_r6p(k)
293      &           -c10*cvh3*r2*u_r4p(k)
294      &            +c5*cvh4*r1*u_r2p(k)
295      &            -cvh5*u_r(k))
296 c p=3
297      &  -cc*c1_322560*(          r7*u_r14p(k)
298      &                 -c7*cvh1*r6*u_r12p(k)
299      &                +c21*cvh2*r5*u_r10p(k)
300      &                -c35*cvh3*r4*u_r8p(k)
301      &                +c35*cvh4*r3*u_r6p(k)
302      &                -c21*cvh5*r2*u_r4p(k)
303      &                 +c7*cvh6*r1*u_r2p(k)
304      &                    -cvh7*u_r(k))
305       enddo
306
307 c Fill in values for the uncmputable points
308         tnh1=2.0D0*(n+0.5D0)*dt*pi2
309         u_i(1)=-sin(tnh1)*sin(pi*x(1))
310         u_i(2)=-sin(tnh1)*sin(pi*x(2))
311         u_i(KE-1)=-sin(tnh1)*sin(pi*x(KE-1))
```

```
312         u_i(KE-2)=-sin(tnh1)*sin(pi*x(KE-2))
313
314 c calculate the exact solution, and determine
315 c the max error for this iteration
316       emax_r=0.0D0
317       emax_i=0.0D0
318       do k=k_start,k_end
319         e_r = dcos(tn1)*dsin(pi*x(k))
320         e_i = -dsin(tnh1)*dsin(pi*x(k))
321
322         err_r = dabs(u_r(k)-e_r)
323         err_i = dabs(u_i(k)-e_i)
324
325         if(emax_r .le. err_r) then
326           emax_r = err_r
327         endif
328
329         if(emax_i .le. err_i) then
330           emax_i = err_i
331         endif
332       enddo
333 c determine the max error, this could be rewritten using the max
      ()
334 c intrinsic function
335       if(emax_r .le. emax_i) then
336         error_max(n) = emax_i
337       else
338         error_max(n) = emax_r
339       endif
340 c print the largest error obtained within this timestep
341 c the conditionals may be removed, the logic ensures that
342 c no more than 32000 lines are ouputed, which makes plotting
343 c and file size much smaller when working with very small mesh
344 c ratios. skipN may be changed at the start of the program if
345 c one wishes more or fewer lines of output
346       if(nsteps .lt. 32000) then
347         print *, n,',', n*dt,',', error_max(n)
348       else
349         if(mod(n,skipN) .eq. 0) then
350           print *, n,',', n*dt,',', error_max(n)
351         endif
352       endif
353
354 c end of the time loop
355       enado
356
357 c determine the largest error observed in the entire simulation
358       err_max=0.0D0
359       do n=1,nsteps
360         if(err_max.LE.error_max(n)) then
361           err_max=error_max(n)
362           n_max=n
363         endif
```

```
364       enddo
365       print *, 'Largest Error: ', n_max, err_max
366 c write all errors, and assocaited time intervals to a file
367 c this may be commented out, if one is running the program from
368 c the command line and piping the output in a file
369       open (unit=22,file='error_max_FDTD_N_3_0_6_x_200_ra_1.00.
             dat')
370       do  n=1,nsteps
371         write(22,10)n*dt, error_max(n)
372       enddo
373       close(22)
374   10  format(f10.8,1x,F20.16)
375
376 c This is the end of the main program
377       stop
378       end
379
380 c This routine computes the 2nd derivaitve
381 c input: f - the function to differentiate
382 c input: KE - the number of grid points, indexed from zero
383 c output: f" in the array fdp
384       subroutine calsed(f,fdp,KE)
385       implicit none
386       dimension f(0:KE),fdp(0:KE)
387       double precision f,fdp
388       double precision a0,a1,a2,a3
389       integer k,KE
390
391 c Compute the 2nd derivative using a sixth-order accurate
392 c Central Difference
393       a0= 49D0/18D0
394       a1= -1.5D0
395       a2= 0.15D0
396       a3= -1.0D0/90.0D0
397       do k=3,KE-3
398
399       fdp(k)= -(a3*f(k-3)+a2*f(k-2)+a1*f(k-1)
400      &          +a0*f(k)+a1*f(k+1)+a2*f(k+2)+a3*f(k+3))
401       enddo
402 c This is the end of the subroutine calsed
403       return
404       end
```

Listing C.2: Model problem using the sixth-order accurate scheme

C.3 Particle Simulation Using Sixth-Order Accurate Scheme

```
1 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
2 c Electron in 1D Free Space Simulation
3 c
4 c This code utilizes a sixth-order accurate central difference
5 c approximation of the Laplace operator. By default the program
6 c utilizes the parameter N=3 for the Generalized FDTD Scheme.
7 c One must comment out the appropriate sections of code should
8 c a lower N value be desired
9 c
10 c This code was adapated from code written by
11 c       Weizhong Dai and Fred Moxley (c) 2011
12 c
13 c This code was modified by James Elliott (c) 2011
14 c
15 c This is the beginning of the main program
16 c all variables must be declared before any assignments are made
17 c parameter values may only be assigned here, and may never be
18 c changed (they are constants)
19 c
20 c ********************* IMPORTANT *********************
21 c ALWAYS write real values using ###.###D##
22 c "D" ensures the values will be double precision
23 c alternatively, when compiling the program, utilize the flags
24 c gfortran -O -fdefault-real-8 -fdefault-double-8 -frange-check -
     Wall
25 c this will ensure all real values are double precision,
26 c as well as enable useful warnings such as unused variables or
27 c loops running past an array's limits
28 c ********************* IMPORTANT *********************
29       implicit none
30
31       integer KE
32       double precision ddx
33 c These parameters are defined by Dai, and should not be changed
34 c Input. KE - the number of spatial grid points (indexed from
     zero)
35 c Input. ddx - the spatial step for the x direction
36       parameter(ddx=1.0D-11, KE=1600)
37 c Declare the size of the arrays
38 c u_r is an array for the real component of the wavefunction
39 c u_i is an array for the imaginary component of the wavefunction
40 c u_r#p corresponds to an #th derivative of the real component
41 c u_i#p corresponds to an #th derivative of the imaginary
     component
42       dimension vp(0·KE),
43      & u_r(0:KE),  u_r2p(0:KE), u_r4p(0:KE), u_r6p(0:KE),
44      & u_r8p(0·KE),u_r10p(0:KE),u_r12p(0:KE),u_r14p(0:KE),
45      & u_i(0:KE),  u_i2p(0:KE), u_i4p(0·KE), u_i6p(0:KE),
46      & u_i8p(0.KE),u_i10p(0:KE),u_i12p(0:KE),u_i14p(0:KE)
47
48 c Declare the type for each variable
49       double precision
```

```
50      & u_r, u_r2p, u_r4p, u_r6p, u_r8p, u_r10p, u_r12p, u_r14p,
51      & u_1, u_12p, u_14p, u_16p, u_18p, u_110p, u_112p, u_114p,
52      & vp, vpot, pi, clambda, sigma, hbar, ptot, ra,
53      $ clap_r, clap_1, cke_r, cke_1, cmelec, ccl, PE,
54      & cvh1, cvh2, cvh3, cvh4, cvh5, cvh6, cvh7,
55      &ch2m1,ch2m2,ch2m3,ch2m4,ch2m5,ch2m6,ch2m7,
56      & r1,r2,r3,r4,r5,r6,r7,
57      & cc, ptotSQRT,mu,hbar2,dt
58
59       integer k,kstart,n,nsteps,kcenter, stop1, stop2
60
61 c These values are the coefficients of the taylor series
62       double precision c1_24,c3,c5,c10,c21,c35,c7,c1_322560,
            c1_1920
63 c These values are the coefficients of the central difference for
64 c energy calculations.
65       double precision a0,a1,a2,a3,dtb
66
67       pi=3.141592653589793238 46D0
68 c the mass of an electron
69       cmelec=9.2D-31
70 c Reduced Planck's constant
71       hbar=1.055D-34
72       hbar2 = hbar*hbar
73       mu=0.98D0
74 c I use GNU Maxima to solve my stability condition in terms of ra
     ,
75 c provided I supply dx, hbar, max|V|, and m.
76 c For comparison this mesh ration should mat Moxely and Dai's mu
77 c So the times are roughly the same
78       ra=20600.0D0
79 c dtb is the original dt with stability condition as written
80 c by Dai and Moxley, I use their dt and (mu) to compute roughly
81 c the same timestep so I may compare against them
82       dtb=2.0D0*(1000.0D0*9.2D0/1.055D0)*ddx*ddx*mu
83 c Because I solve for a specific mesh ratio, I may calculate my
84 c timestep in the same manner as the exact problem.
85       dt = ddx*ddx*ra
86 c These timesteps are roughly the same as Moxley and Dai, it may
     be
87 c required to add or subtract some small amount to make them
     closer
88       stop2 = int((dtb*1300D0)/dt) -150
89       stop1 = int((dtb*350D0)/dt) - 100
90       nsteps =stop2
91       r1=dt/(ddx*ddx)
92       r2=r1*r1
93       r3=r2*r1
94       r4=r3*r1
95       r5=r4*r1
96       r6=r5*r1
97       r7=r6*r1
98 c these values are scaled from (h/2m)
```

```
 99        ch2m1=1.055D0/(2000.0D0*9.2D0)
100        ch2m2=ch2m1*ch2m1
101        ch2m3=ch2m2*ch2m1
102        ch2m4=ch2m3*ch2m1
103        ch2m5=ch2m4*ch2m1
104        ch2m6=ch2m5*ch2m1
105        ch2m7=ch2m6*ch2m1
106 c  These values define the Gaussian packet
107        clambda=10.0D0
108        sigma=10.0D0
109 c V_0 is 100 eV
110        vpot=100.0D0
111 c These values defined the initial location of the packet, and
112 c where the potential begins
113        kstart=800
114        kcenter=400
115        cc=1.0D0
116
117 c Coefficients in the taylor expansion, to ensure double
       precision
118        c1_24=1.0D0/24.0D0
119        c3=3.0D0
120        c5=5.0D0
121        c10=10.0D0
122        c21=21.0D0
123        c35=35.0D0
124        c7=7.0D0
125        c1_322560=1.0D0/322560.0D0
126        c1_1920=1.0D0/1920.0D0
127 c sixth-order accurate central difference coefficients
128        a0= 49D0/18D0
129        a1= -1.5D0
130        a2= 0.15D0
131        a3= -1.0D0/90.0D0
132 c This out helps to realize the relation between a timestep in
       this
133 c simulation, and a timestep in Dai and Moxley's simulation
134        print *, dt, dtb , nsteps, dtb*1300D0/dt
135        print *, '350 = ', stop1, ', 1300 = ', stop2
136 c This code is parallelized using OMP, though it will only be
       used
137 c if you enable the compile flag -fopenmp
138 c This applies the initial conditions
139 c$OMP PARALLEL DO
140 c$OMP& SCHEDULE (static)
141        do k=0,KE
142        u_r(k)=0.0
143        u_i(k)=0.0
144        vp(k)=0.0
145        enddo
146 c$OMP END PARALLEL DO
147 c In this case eV is converted to Joules
148 c$OMP PARALLEL DO
```

```
149 c$OMP& SCHEDULE (static)
150       do k=kstart,KE-10
151       vp(k)=vpot*1.602D-19
152       enddo
153 c$OMP END PARALLEL DO
154
155       ptot=0.0
156 c Initiate the Gaussian packet
157 c$OMP PARALLEL DO REDUCTION(+:ptot)
158 c$OMP& SCHEDULE (static)
159       do k=10,kstart-10
160       u_r(k)=dcos(2.0D0*pi*(k-kcenter)/clambda)
161      $     *dexp(-0.5D0*((k-kcenter)/sigma)**2)
162       u_i(k)=dsin(2.0D0*pi*(k-kcenter)/clambda)
163      $     *dexp(-0.5D0*((k-kcenter)/sigma)**2)
164       ptot=ptot+(u_r(k)**2)+(u_i(k)**2)
165       enddo
166 c$OMP END PARALLEL DO
167
168       ptotSQRT = sqrt(ptot)
169 c Compute the initial values of the wavefunction
170 c (this is taken from Sullivan)
171 c$OMP PARALLEL DO
172 c$OMP& SCHEDULE (static)
173       do k=10,kstart-10
174       u_r(k)=u_r(k)/ptotSQRT
175       u_i(k)=u_i(k)/ptotSQRT
176       enddo
177 c$OMP END PARALLEL DO
178
179 c Compute the energy in the system, second, and fourth-order
      accurate
180 c derivatives are provided, but sixth-order is the default
181       ccl=0.5D0*1.055D0*1.055D0*6.23D0*1000D0/9.2D0
182       cke_r=0.0
183       cke_i=0.0
184       PE=0.0
185
186 c$OMP PARALLEL DO PRIVATE(clap_r,clap_i)
187 c$OMP& REDUCTION(+:PE) REDUCTION(+:cke_r) REDUCTION(+:cke_i)
188       do k=10,KE-10
189 c       clap_r=u_r(k+1)-2.0*u_r(k)+u_r(k-1)
190 c       clap_i=u_i(k+1)-2.0*u_i(k)+u_i(k-1)
191
192         clap_r=-(a3*u_r(k-3)+a2*u_r(k-2)+a1*u_r(k-1)
193      &          +a0*u_r(k)+a1*u_r(k+1)+a2*u_r(k+2)+a3*u_r(k+3))
194
195         clap_i=-(a3*u_i(k-3)+a2*u_i(k-2)+a1*u_i(k-1)
196      &          +a0*u_i(k)+a1*u_i(k+1)+a2*u_i(k+2)+a3*u_i(k+3))
197
198 c       clap_r=(-u_r(k+2)+16.0D0*u_r(k+1)-30.0D0*u_r(k)+16.0D0*u_r
      (k-1)
199 c     &           -u_r(k-2))/12.0D0
```

```
200 c       clap_i=(-u_i(k+2)+16 0D0*u_i(k+1)-30 0D0*u_i(k)+16 0D0*u_i
      (k-1)
201 c      &           -u_i(k-2))/12 0D0
202          cke_r=cke_r+u_r(k)*clap_r+u_i(k)*clap_i
203          cke_i=cke_i+u_r(k)*clap_i-u_i(k)*clap_r
204          PE=PE+vp(k)*((u_r(k)**2)+(u_i(k)**2))
205        enddo
206 c$OMP END PARALLEL DO
207 c output the potential and kinetic energy
208        print *, 0, ccl*sqrt(cke_r**2+cke_i**2), PE*1 0D+19/1 602D0
             ,
209       &  ccl*sqrt(cke_r**2+cke_i**2) + PE*1 0D+19/1 602D0
210 c save the intial energy to a csv file
211        open(unit=22,file='psi_im-n-0 csv )
212        do  k=1,KE-1
213          write(22,10)k, u_i(k)
214        endo
215        close(22)
216        oper(unit=1,file='psi_rl-n-0 csv')
217        do k=1,KE-1
218          write(1,10)k, u_r(k)
219        endo
220        close(1)
221
222        do n=1,nsteps
223
224 c calculate the derivatives of imaginary values
225        call calsed(u_i,u_i2p,KE)
226        call calsed(u_i2p,u_i4p,KE)
227        call calsed(u_i4p,u_i6p,KE)
228        call calsed(u_i6p,u_i8p,KE)
229        call calsed(u_i8p,u_i10p,KE)
230        call calsed(u_i10p,u_i12p,KE)
231        call calsed(u_i12p,u_i14p,KE)
232
233 c$OMP PARALLEL DO PRIVATE(cvh1,cvh2,cvh3,cvh4,cvh5,cvh6,cvh7)
234 c$OMP& SCHEDULE (static)
235        do k=10,KE-10
236        cvh1=(vp(k)*dt)/hbar
237        cvh2=cvh1*cvh1
238        cvh3=cvh2*cvh1
239        cvh4=cvh3*cvh1
240        cvh5=cvh4*cvh1
241        cvh6=cvh5*cvh1
242        cvh7=cvh6*cvh1
243
244 c calculate the real values
245        u_r(k)=u_r(k)-r1*ch2m1*u_i2p(k)+cvh1*u_i(k)
246 c p=1
247       $    +c1_24*(          ch2m3*r3*u_i6p(k)
248       &           -c3*cvh1*ch2m2*r2*u_i4p(k)
249       &           +c3*cvh2*ch2m1*r1*u_i2p(k)
250       &              -cvh3*u_i(k))
```

```
251 c p=2
252      &  -c1_1920*(           ch2m5*r5*u_i10p(k)
253      &               -c5*cvh1*ch2m4*r4*u_i8p(k)
254      &              +c10*cvh2*ch2m3*r3*u_i6p(k)
255      &              -c10*cvh3*ch2m2*r2*u_i4p(k)
256      &               +c5*cvh4*ch2m1*r1*u_i2p(k)
257      &                  -cvh5*u_i(k))
258 c p=3
259      &  +cc*c1_322560*(          ch2m7*r7*u_i14p(k)
260      &                  -c7*cvh1*ch2m6*r6*u_i12p(k)
261      &                 +c21*cvh2*ch2m5*r5*u_i10p(k)
262      &                 -c35*cvh3*ch2m4*r4*u_i8p(k)
263      &                 +c35*cvh4*ch2m3*r3*u_i6p(k)
264      &                 -c21*cvh5*ch2m2*r2*u_i4p(k)
265      &                  +c7*cvh6*ch2m1*r1*u_i2p(k)
266      &                     -cvh7*u_i(k))
267
268       enddo
269 c$OMP END PARALLEL DO
270
271 c calculate the derivatives of real values
272       call calsed(u_r,u_r2p,KE)
273       call calsed(u_r2p,u_r4p,KE)
274       call calsed(u_r4p,u_r6p,KE)
275       call calsed(u_r6p,u_r8p,KE)
276       call calsed(u_r8p,u_r10p,KE)
277       call calsed(u_r10p,u_r12p,KE)
278       call calsed(u_r12p,u_r14p,KE)
279
280 c$OMP PARALLEL DO PRIVATE(cvh1,cvh2,cvh3,cvh4,cvh5,cvh6,cvh7)
281 c$OMP& SCHEDULE (static)
282       do k=10,KE-10
283       cvh1=(vp(k)*dt)/hbar
284       cvh2=cvh1*cvh1
285       cvh3=cvh2*cvh1
286       cvh4=cvh3*cvh1
287       cvh5=cvh4*cvh1
288       cvh6=cvh5*cvh1
289       cvh7=cvh6*cvh1
290
291 c calculate the imaginary values
292       u_i(k)=u_i(k)+ch2m1*r1*u_r2p(k)-cvh1*u_r(k)
293 c p=1
294      $    -c1_24*(          ch2m3*r3*u_r6p(k)
295      &            -c3*cvh1*ch2m2*r2*u_r4p(k)
296      &            +c3*cvh2*ch2m1*r1*u_r2p(k)
297      &               -cvh3*u_r(k))
298 c p=2
299      &  +c1_1920*(           ch2m5*r5*u_r10p(k)
300      &               -c5*cvh1*ch2m4*r4*u_r8p(k)
301      &              +c10*cvh2*ch2m3*r3*u_r6p(k)
302      &              -c10*cvh3*ch2m2*r2*u_r4p(k)
303      &               +c5*cvh4*ch2m1*r1*u_r2p(k)
```

```
304      &                  -cvh5*u_r(k))
305 c p=3
306      &  -cc*c1_322560*(           ch2m7*r7*u_r14p(k)
307      &                  -c7*cvh1*ch2m6*r6*u_r12p(k)
308      &                 +c21*cvh2*ch2m5*r5*u_r10p(k)
309      &                 -c35*cvh3*ch2m4*r4*u_r8p(k)
310      &                 +c35*cvh4*ch2m3*r3*u_r6p(k)
311      &                 -c21*cvh5*ch2m2*r2*u_r4p(k)
312      &                  +c7*cvh6*ch2m1*r1*u_r2p(k)
313      &                     -cvh7*u_r(k))
314       enddo
315 c$OMP END PARALLEL DO
316
317
318 c Compute the energy in the system, second, and fourth-order
     accurate
319 c derivatives are provided, but sixth-order is the default
320       cke_r=0.0
321       cke_i=0.0
322       PE=0.0
323
324 c$OMP PARALLEL DO PRIVATE(clap_r,clap_i)
325 c$OMP& REDUCTION(+:PE) REDUCTION(+:cke_r) REDUCTION(+:cke_i)
326       do k=10,KE-10
327 c        clap_r=u_r(k+1)-2.0*u_r(k)+u_r(k-1)
328 c        clap_i=u_i(k+1)-2.0*u_i(k)+u_i(k-1)
329         clap_r=-(a3*u_r(k-3)+a2*u_r(k-2)+a1*u_r(k-1)
330      &          +a0*u_r(k)+a1*u_r(k+1)+a2*u_r(k+2)+a3*u_r(k+3))
331
332         clap_i=-(a3*u_i(k-3)+a2*u_i(k-2)+a1*u_i(k-1)
333      &          +a0*u_i(k)+a1*u_i(k+1)+a2*u_i(k+2)+a3*u_i(k+3))
334
335 c       clap_r=(-u_r(k+2)+16.0D0*u_r(k+1)-30.0D0*u_r(k)+16.0D0*u_r
     (k-1)
336 c      &           -u_r(k-2))/12.0D0
337 c       clap_i=(-u_i(k+2)+16.0D0*u_i(k+1)-30 0D0*u_i(k)+16.0D0*u_i
     (k-1)
338 c      &           -u_i(k-2))/12.0D0
339         cke_r=cke_r+u_r(k)*clap_r+u_i(k)*clap_i
340         cke_i=cke_i+u_r(k)*clap_i-u_i(k)*clap_r
341         PE=PE+vp(k)*((u_r(k)**2)+(u_i(k)**2))
342       enddo
343 c$OMP END PARALLEL DO
344 c output the energy in the system at this time
345       if(mod(n,10) .eq. 0) then
346       print *, n, ccl*sqrt((cke_r**2)+(cke_i**2)), PE*1 0D
            +19/1 602D0,
347      &  ccl*sqrt(cke_r**2+cke_i**2) + PE*1 0D+19/1.602D0
348       endif
349
350 c save the wavefunctions if the timestep for comparison
351       if(n .eq  stop2) then
352         open (unit=22,file='psi_r-n-1300 csv')
```

```
353         do k=1,KE-1
354         write(22,10)k, u_i(k)
355         enddo
356         close(22)
357
358         open (unit=1,file='psi_rl-n-1300.csv')
359         do k=1,KE-1
360         write(1,10)k, u_r(k)
361         enddo
362         close(1)
363 c ensure the energy is printed (it may not because of the
364 c modulus function above)
365       print *, n, ccl*sqrt((cke_r**2)+(cke_i**2)), PE*1.0D
              +19/1.602D0,
366      &  ccl*sqrt(cke_r**2+cke_i**2) + PE*1.0D+19/1.602D0
367
368       else if(n .eq. stop1) then
369         open (unit=22,file='psi_im-n-350.csv')
370         do k=1,KE-1
371         write(22,10)k, u_i(k)
372         enddo
373         close(22)
374
375         open (unit=1,file='psi_rl-n-350.csv')
376         do k=1,KE-1
377         write(1,10)k, u_r(k)
378         enddo
379         close(1)
380 c ensure the energy is printed (it may not because of the
381 c modulus function above)
382       print *, n, ccl*sqrt((cke_r**2)+(cke_i**2)), PE*1.0D
              +19/1.602D0,
383      &  ccl*sqrt(cke_r**2+cke_i**2) + PE*1.0D+19/1.602D0
384       endif
385 c This ends the time loop
386       enddo
387
388   10  format(I5,',',F25.15)
389 c This ends the program
390       stop
391       end
392
393
394 c This routine computes the 2nd derivaitve
395 c input. f - the function to differentiate
396 c input: KE - the number of grid points, indexed from zero
397 c output: f" in the array fdp
398       subroutine calsed(f,fdp,KE)
399       implicit none
400       dimension f(0:KE),fdp(0:KE)
401       double precision f,fdp,a0,a1,a2,a3
402       integer k,ke
403
```

```
404 c    Six-order Scheme
405        a0= 49.0D0/18.0D0
406        a1= -1.5D0
407        a2= 0.15D0
408        a3= -1.0D0/90.0D0
409
410 c$OMP PARALLEL DO
411 c$OMP& SCHEDULE (static)
412        do k=10,KE-10
413        fdp(k)= -(a3*f(k-3)+a2*f(k-2)+a1*f(k-1)
414       &          +a0*f(k)+a1*f(k+1)+a2*f(k+2)+a3*f(k+3))
415        enddo
416 c$OMP END PARALLEL DO
417
418 c clamp the ends to zero
419 c an absorbing boundary condition should be here!
420        do k=0,9
421          fdp(k)=0.0
422        enddo
423
424        do k=KE-9,KE
425          fdp(KE-k)=0.0
426        enddo
427
428        return
429        end
```

Listing C.3: Particle simulation using the sixth-order accurate scheme

BIBLIOGRAPHY

[1] A. Arnold, M. Ehrhardt, and I. Sofronov, *Discrete transparent boundary conditions for the Schrödinger equation: Fast calculation, approximation and stability, communications in mathematical*, Science, 1 (2003), pp. 501–556.

[2] A. Askar and A. S. Cakmak, *Explicit integration method for the time-dependent Schrödinger equation*, J. Chem. Phys., 68 (1978), pp. 2794–2798.

[3] K. Atkinson, *An Introduction to Numerical Analysis*, John Wiley & Sons, New York, 1989.

[4] A. Bao, S. Jin, and P. Markowich, *On time-splitting spectral approximations for the Schrödinger equation in the semiclassical regime*, J. Comput. Phys., 26 (2005), pp. 487–524.

[5] N. Carjan, M. Rizea, and S. Strottman, *Efficient numerical solution of the time-dependent Schrödinger equation for deep tunneling*, Romanian Reports in Physics, 55 (2003), pp. 555–579.

[6] H. Chen and B. Shizgal, *The quadrature discretization method in the solution of the Schrödinger equation*, J. Chem., 24 (1998), pp. 321–343.

[7] Z. Chen, J. Zhang, and Z. Yu, *Solution of the time-dependent Schrödinger equation with absorbing boundary conditions*, Journal of Semiconductors, 30 (2009), p. 012001.

[8] G. Dahlquist, *Numerical Methods*, Dover Publications, New Jersey, 2003.

[9] W. Dai, *An unconditionally stable three-level explicit difference scheme for the Schrodinger equation with a variable coefficient*, SIAM J. Numer. Anal., 29 (1992), pp. 174–181.

[10] W. Dai, G. Li, R. Nassar, and S. Su, *On the stability of the FDTD method for solving a time-dependent Schrödinger equation*, Numer. Methods Partial Differential Eq., 21 (2005), pp. 1140–1154.

[11] W. Dai and F. Moxley, *A generalized finite difference time domain method for solving a time dependent linear Schrödinger equation*, Submitted, (2011).

[12] S. Descomb and T. M., *An exact local error representation of exponential operator splitting methods for evolutionary problems and applications to linear Schrödinger equations in the semi-classical regime*, BIT Numer. Math., 50 (2010), pp. 729–749.

[13] T. Fevens and H. Jiang, *Absorbing boundary conditions for the Schrödinger equation*, SIAM J. Sci. Comput., 21 (1999), pp. 255–282.

[14] F. Fujiwara, *FDTD-Q analysis of a Mott insulator quantum phase*, PhD thesis, University of Tokyo, Japan, 2007.

[15] H. Han, J. Jin, and X. Wu, *A finite-difference method for the one dimensional time dependent Schrödinger equation on unbounded domain*, Computers and Mathematics with Applications, 50 (2005), pp. 1345–1362.

[16] M. Hochbruuck and C. Lubich, *On Magnus integrators for time-dependent Schrödinger equations*, SIAM J. Numer. Anal., 41 (2003), pp. 945–963.

[17] B. Jazbi and M. Moini, *On the numerical solution of one dimensional Schrödinger equation with boundary conditions involving fractional differential operators*, IUST International Journal of Engineering Science, 19 (2008), pp. 21–26.

[18] Z. Kalogiratou, T. Monovasilis, and T. E. Simos, *Asymptotically symplectic integrators of 3rd and 4th order for the numerical solution of the Schrödinger equation*, Computational Fluid and Solid Mechanics, (2003), pp. 2012–2015.

[19] K. S. Kunz and R. J. Luebbers, *The finite difference time domain method for electromagnetics*, CRC Press, Florida, 1993.

[20] J. P. Kuska, *Absorbing boundary conditions for the Schrödinger equation on finite intervals*, Physcial Review B, 46 (1992), pp. 5000–5003.

[21] J. Lo and B. D. Shizgal, *Spectral convergence of the quadrature discretization method in the solution of the Schrödinger and fokker-planck equations: comparison with sinc methods*, J. Chem. Phys., 125 (2006), p. 194108.

[22] K. W. Morton and D. F. Mayers, *Numerical solution of partial differential equations*, Cambridge University Press, London, 1994.

[23] P. L. Nash and L. Y. Chen, *Efficient finite difference solutions to the time-dependent Schrödinger equation*, J. Comput. Phys., 130 (1997), pp. 266–268.

[24] J. Quan and C. T. Chang, *New insights in solving distributed system equations by the quadrature method–ii. numerical experiments*, Computers & Chemical Engineering, 13 (1989), pp. 1017 – 1024.

[25] J. R. Quan and C. T. Chang, *New insights in solving distributed system equations by the quadrature method–i. numerical experiments*, Computers & Chemical Engineering, 13 (1989), pp. 779 – 788.

[26] R. F. Remis, *On the relation between FDTD and fibonacci polynomials*, Journal of Computational Physics, 230 (2010), pp. 1382–1386.

[27] C. Shu, *Differential Quadrature and Its Application in Engineering*, Springer, London, 2000.

[28] A. Soriano, E. A. Navarro, J. A. Porti, and V. Such, *Analysis of the finite difference time domain technique to solve the Schrodinger equation for quantum devices*, Journal of Applied Physics, 95 (2004), pp. 8011–8018.

[29] D. M. Sullivan, *Electromagnetic simulation using the FDTD method*, IEEE Press, New York, 2000.

[30] D. M. Sullivan and D. Citrin, *Time-domain simulation of a universal quantum gate*, Journal of Applied Physics, 96 (2002), pp. 3219–3226.

[31] D. M. Sullivan and D. S. Citrin, *Time-domain simulation of two electrons in a quantum dot*, Journal of Applied Physics, 89 (2001), pp. 3841–3846.

[32] J. Szeftel, *Design of absorbing boundary condition for Schrödinger equations in R^d*, SIAM J. Numer. Anal., 42 (2004), pp. 1527–1551.

[33] L. Wu, *Dufort-Frankel-type methods for linear and nonlinear Schrödinger equations*, SIAM J. Numer. Anal., 33 (1996), pp. 1526–1533.

[34] S. Zhao and G. W. Wei, *High-order FDTD methods via derivative matching for Maxwell's equations with material interfaces*, Journal of Comput. Phy., 200 (2004), pp. 60–103.