

Spring 2006

Dynamics of Nonlinear Diffusion Processes

Dustin A. Sipka

Embry-Riddle Aeronautical University - Daytona Beach

Follow this and additional works at: <https://commons.erau.edu/db-theses>



Part of the [Aerospace Engineering Commons](#), and the [Physics Commons](#)

Scholarly Commons Citation

Sipka, Dustin A., "Dynamics of Nonlinear Diffusion Processes" (2006). *Theses - Daytona Beach*. 186.
<https://commons.erau.edu/db-theses/186>

This thesis is brought to you for free and open access by Embry-Riddle Aeronautical University – Daytona Beach at ERAU Scholarly Commons. It has been accepted for inclusion in the Theses - Daytona Beach collection by an authorized administrator of ERAU Scholarly Commons. For more information, please contact commons@erau.edu.

DYNAMICS OF NONLINEAR DIFFUSION
PROCESSES

By

Dustin A. Sipka

A thesis Submitted to the
Physical Sciences Department
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Space Science

Embry-Riddle Aeronautical University
Daytona Beach, Florida
Spring 2006

UMI Number: EP32095

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform EP32095
Copyright 2011 by ProQuest LLC
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Copyright by Dustin Andrew Sipka 2006
All Rights Reserved

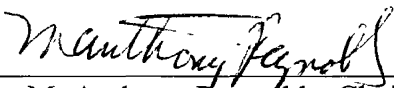
DYNAMICS OF NONLINEAR DIFFUSION
PROCESSES

by

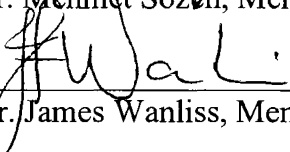
Dustin A. Sipka

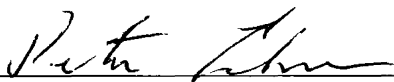
This thesis was prepared under the direction of the candidate's thesis committee chair, Dr. M. Anthony Reynolds, Department of Physical Sciences, and has been approved by the members of his thesis committee. It was submitted to the Department of Physical Sciences and was accepted in partial fulfillment of the requirements for the Degree of Master of Science in Space Science

THESIS COMMITTEE:


Dr. M. Anthony Reynolds, Chair


Dr. Mehmet Sozen, Member


Dr. James Wanliss, Member


Dr. Peter Erdman, MSSPS Graduate Program Coordinator


Dr. John Olivero, Department Chair, Physical Sciences


Dr. John Watret, Associate Provost

4-27-06
Date

ACKNOWLEDGEMENTS

To my family and friends. Without all of you I would not be the person I am today. A special thanks goes to my parents who instilled in me the values of perseverance and hard work without which I would not have been able to complete this endeavor.

To my wonderful fiancée, Shari Lynn DiSalvo, who has shown me another side of life that I have never known. Your support and encouragement has been tremendous. Thank you for taking this journey with me.

ABSTRACT

Author: Dustin A. Sipka
Title: Dynamics of Nonlinear Diffusion Processes
Institution: Embry-Riddle Aeronautical University
Degree: Master of Science in Space Science
Year: 2006

The purpose of this thesis is to analyze nonlinear diffusion processes. In particular, some of the results arrived at by Newman and Sagan in their 1981 paper “Galactic Civilizations: Population Dynamics and Interstellar Diffusion,” will be reproduced by different means. First, a thorough analysis of the linear diffusion equation will be performed in order to test a numerical algorithm that can solve the nonlinear diffusion equation and look at the processes of interest with sufficient accuracy. Once the algorithm is tested and shows good resolution it is used to solve the nonlinear equation. The post processing is then done to compare the numerical results with the analytical solution and then they are related to the results arrived at by Newman and Sagan. More precisely, the timescales over which these processes take place are of great interest. A study of the dynamics of these diffusion processes that take place will bring about a better understanding of the nature of nonlinear diffusion and some of its applications.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	iv
ABSTRACT.....	v
TABLE OF CONTENTS.....	vi
LIST OF TABLES.....	ix
LIST OF FIGURES.....	x
1. Introduction.....	1
1.1 Synopsis.....	5
2. Background.....	10
2.1 Diffusion.....	10
2.2 Nonlinear Science.....	11
2.3 Plasmas.....	13
2.4 Population Expansion.....	14
3. The Linear Diffusion Equation.....	16
3.1 Modeling the Linear Diffusion Equation.....	16
3.2 Developing the Code.....	20
3.3 LU Decomposition.....	24
3.4 Crank-Nicolson Method.....	27
3.5 Comparison of the Numerical Results to the Analytic Result.....	32
4. The Nonlinear Diffusion Equation.....	42
4.1 Development of the Nonlinear Diffusion Equation.....	42
4.2 Time Linearization.....	44

4.3	Steady-State	46
4.4	Specific Condition of Interest	49
4.5	Linear Regime of Nonlinear Equation.....	50
4.5.1	Steady-State of Selected Initial Condition.....	50
4.6	Nonlinear Regime	52
4.6.1	Derivation of Theoretical Timescale of Nonlinear Equation.....	52
4.6.2	Results of the Numerical Analysis of the Nonlinear Diffusion Equation.	57
4.6.3	Relation to Interstellar Population Expansion	62
5.	Conclusions.....	66
	References.....	69
	Appendix A: Additional Derivations	71
A.1	Calculating the Stability of the Explicit Method (von Neumann stability analysis)	71
A.2	Adding a Source Term	72
	Appendix B: Code.....	76
B.1	ConstantSourceGoodTimeNonlinearDiffusion.m	76
B.2	FinalAlgorithmNonlinear.m.....	79
B.3	FinalCrankNicolsonAlgorithmNonlinear.m	83
B.4	GoodDiffusion.m	90
B.5	GoodTimeNonlinearDiffusion.m.....	95
B.6	HomoNonlinearDiffusion.m	98
B.7	InitialConditionofInterest.m.....	99
B.8	SourceGoodTimeNonlinearDiffusion.m.....	101

B.9	TimeConstantsNonlinear.m	105
B.10	VaryingParameter.m	107
B.11	VaryingParameter2.m	112
B.12	WavefrontVelocity.m.....	116

LIST OF TABLES

Table 1. Comparison of Lowest Absolute RMS Error for Crank-Nicolson Method..... 31

LIST OF FIGURES

Figure 1. Steady-State Solutions to the Nonlinear Diffusion Equation.	2
Figure 2. Initial Conditions of Interest.....	3
Figure 3. Value of Parameter that Yields Lowest Absolute RMS Error.	29
Figure 4. Initial Waveform used to Verify Code.	34
Figure 5. Time Lapse of Initial Waveform.	35
Figure 6. Decay of Amplitudes of Different Modes of Initial Waveform.	36
Figure 7. Absolute Error Relative to Position.....	37
Figure 8. RMS Error Comparison between Numerical and Exact Solutions.	38
Figure 9. Comparison of Amplitudes of Selected Order Mode.....	39
Figure 10. Absolute Error between Amplitudes of Numerical and Exact Solutions.....	40
Figure 11. Absolute Error between Amplitudes of Numerical and Exact Solutions for Smaller Time-step.....	41
Figure 12. Steady-State Solutions for Various Powers of Nonlinearity.....	48
Figure 13. Initial Condition with $n = 0$	50
Figure 14. Theoretical Prediction of Timescale Compared to Actual Values.....	58
Figure 15. Position of Profile when Timescale is Calculated.....	59
Figure 16. Time History of Mode Amplitudes.	60
Figure 17. Time History of Mode Amplitudes.	61
Figure 18. Self-Similar Solutions of the Expanding Wavefront ($n = 4$).....	62
Figure 19. Position vs. Time Graph in Order to see Velocity of Expansion Wavefront.	63
Figure 20. Steady-State Waveform for $n = 1$	74

Figure 21. Relative Error between Numerical and Exact Steady-State Solutions..... 75

1. Introduction

The easiest way to teach a topic in physics is to make ideal assumptions. Certain parameters are ignored for the sake of pure physics. Ignoring friction and looking at linear systems are just two examples of how society teaches physics topics to students. While this brings ease in understanding, it is incorrect. Nature is not ideal; it is very chaotic. This brings in the topic of nonlinearity. Nature can be very nonlinear. Phenomena in Nature are very complex and do not always occur in nice straight-line patterns. Heat transfer, weather, avalanches, and the interaction with the Earth-Space environment are examples of very nonlinear phenomena. One phenomenon that is of particular interest to me is nonlinear diffusion. I propose to investigate nonlinear diffusion processes, in general, and specifically the timescales over which these processes occur.

The problem will be addressed by means of both analytic approximations as well as numerical code. First, an analytical solution to the linear diffusion equation will be derived to give theoretical predictions of how the numerical code should behave. These analytic predictions will allow a good comparison to the numerical code. A numerical code that solves the general diffusion equation for the linear case is developed and the results are compared to the analytic predictions. The main focus is to see if certain initial

conditions behave as desired when they are modeled in the computer. The condition of interest is related to the steady-state of the nonlinear diffusion equation.

For example, consider a metal bar of unit length whose ends are fixed at (scaled) temperatures 0 and 1, respectively. What is the steady-state temperature profile? Of course, the well-known answer is that it is linear. However, if the thermal diffusion coefficient is not constant, the temperature diffusion equation becomes nonlinear. The form assumed in this thesis is

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x} \left(\delta T^n \frac{\partial T}{\partial x} \right) \quad (1)$$

Where n is a nonlinear parameter.

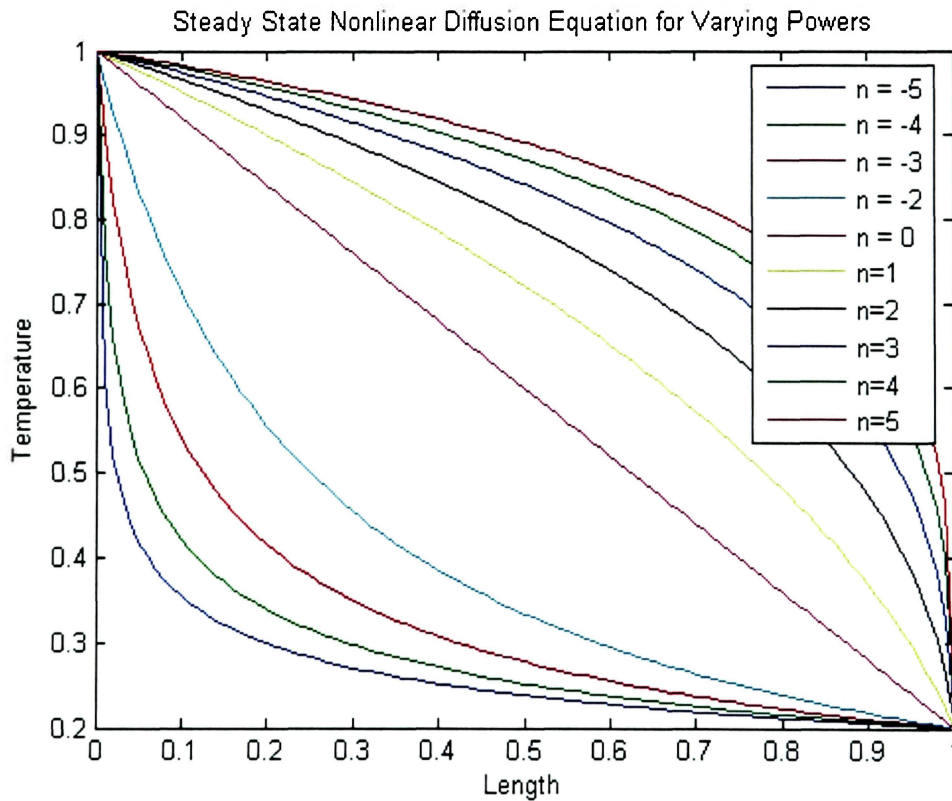


Figure 1. Steady-State Solutions to the Nonlinear Diffusion Equation.

In Figure 1 I show the temperature profiles for various values of n . The case $n = 0$ reproduces the well-known linear result, but the larger the value of $|n|$, the more nonlinear the steady-state temperature profile becomes. In addition to these unusual steady-state profiles, the time history of the approach to equilibrium will be studied. That is, it is of interest to see how long these profiles take to reach equilibrium and the timescale at which they approach their respective final states.

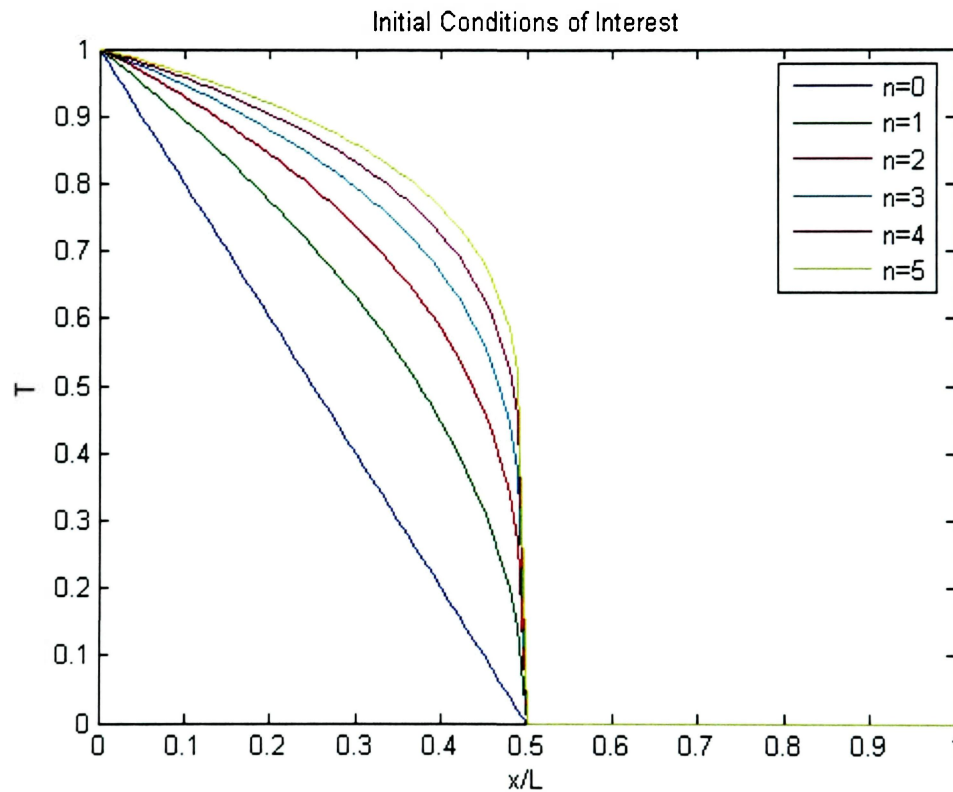


Figure 2. Initial Conditions of Interest.

Of course, one hallmark of nonlinear dynamics is a sensitivity to initial conditions. In the present case, the route to the steady-state profile can be highly dependent on the initial profile. Because one of the physical motivations for this work is to investigate the propagation of a species outward from an initially localized region (as in the case of

humans expanding into the galaxy [Newman and Sagan, 1981]), the initial profile is shown in Figure 2. Of course, since the dependent variable in Equation (1) can be interpreted as population density as well as thermal energy density (i.e., temperature), Equation (1) holds for the latter case.

Because Equation (1) is difficult to solve exactly, a linear approximation of the nonlinear equation will be solved to obtain an approximate analytic solution, which will give theoretical predictions of the timescale over which the diffusion processes occur. To correctly interpret this linear approximation, we first solve the usual linear diffusion equation in great detail, and ultimately apply those results to our approximate solution to the nonlinear equation. After the linear case, our analysis of the nonlinear diffusion equation will investigate key features that are entirely nonlinear in character and have no counterpart in the linear case. Once this is done the numerical results will be compared to the analytical solution of the equation where necessary. A major goal is to compare the numerical results with the results arrived at by Newman and Sagan (1981) in their paper on intergalactic population expansion.

A study of the dynamics of the diffusion processes that take place will bring about a better understanding of the nonlinear nature of diffusion and its applications.

1.1 Synopsis

Chapter 2 describes the motivation for this thesis. The topics of diffusion, nonlinear science, plasmas, and population expansion are discussed in more detail.

Section 2.1 explains different phenomena that can be described by diffusion and goes into mathematical detail of the underlying principles for this work. Equation (2) becomes the basis for all the work in this thesis.

Section 2.2 points out reasons why it is important to study nonlinear phenomena. From hurricane models and predicting the weather it is clear that improvements need to be made on several current models. The Madden-Julian Oscillation [Fell, 2005] is one such phenomenon for which mathematical modeling is used in helping forecast the nonlinear behavior of weather patterns.

Section 2.3 describes another example of an inherently nonlinear phenomenon. This physical situation, described by Equations (6) and (7), is the motivation for the power law dependence of the diffusion coefficient seen in Equation (1).

Section 2.4 gives a brief explanation of factors that contribute to how fast or when a population chooses to expand. Newman and Sagan (1981) give an enlightening treatment of intergalactic population expansion and the methods that will be used in this thesis are described in more detail.

Chapter 3 goes into detailed analysis of the linear diffusion equation (Equation 8). The equation is solved via the method of separation of variables, and then scaled to make the solution unitless. This scaled equation is then used to develop a numerical code that models the behavior of this equation. The results from the numerical code are then compared to the theoretical predictions of the exact analytic solution before the thesis is expanded to the nonlinear case. This chapter is only concerned with developing a numerical code, and can be skipped by the reader who is interested only in the nonlinear results.

Section 3.1 goes through the separation of variables technique utilized to solve Equation (8) and arrives at the exact analytic solution, Equation (18). The process of “scaling” is then explained to set up the comparison to the scaled equation (Equation 27) that the numerical code will model.

Section 3.2 explains the finite differencing techniques that are utilized in this thesis to model the diffusion equation. The spatial and temporal grid is explained along with the von Neumann stability analysis, Equation (38), used to ensure a viable numerical solution. The section then sets up the system of equations that will be solved via the Crank-Nicolson method.

Section 3.3 is a brief explanation of the method that is the basis for the numerical code. This method provides a fast means to calculating the matrix inverse which is necessary to solve the system of equations described by Equation (41).

Section 3.4 is a different means at solving the diffusion equation. It provides an

average of the explicit and implicit methods explained in **Section 3.2**. A brief analysis of Equation (54) is performed to see if the average chosen by Crank and Nicolson provides the lowest error or if a different percentage of the two methods, explicit and implicit, yield a lower error. The results of which are shown in Figure 3.

Section 3.5 compares the numerical results to the analytic result. It provides support for the code to be expanded to the nonlinear case. Figure 6 shows that the modes decay according to Equation (19) and that the error in the solution decreases when Δt decreases (comparison of Figure 10 and Figure 11).

Chapter 4 provides the analysis of the nonlinear diffusion equation. It begins with the development of Equation (58) and transforms it into a finite difference equation that can be modeled numerically. It then goes through the time linearization of the equation that makes all of the results easier to calculate. After that it presents the steady-state case of the equation which is used to simulate population expansion and the results are then analyzed.

Section 4.1 develops the nonlinear diffusion equation (Equation 58) and transforms it into a finite difference equation that can be modeled numerically (Equation 64).

Section 4.2 explains the tool of time linearization. This assumption (Equation

66) ensures that the difference in temperature from one time step to the next is so small that it can be assumed to behave linearly. It performs a Taylor expansion on the dependent variable T with respect to time. This provides ease in computation.

Section 4.3 provides the steady-state solution to the nonlinear diffusion equation (Equation 80). Several profiles are provided in [Figure 12](#).

Section 4.4 takes the steady-state solution from Section 4.3 and allows it to expand into open space, [Figure 13](#).

Section 4.5 looks at the linear regime of the nonlinear equation (i.e., when $n = 0$).

Section 4.5.1 solves for the exact analytic solution to the initial condition described in **Section 4.4**. The solution subtracts out the known steady-state profile to allow for easier calculation of the time-constants of interest.

Section 4.6 looks at the nonlinear regime of the nonlinear diffusion equation (Equation 58).

Section 4.6.1 derives the theoretical prediction of the timescales, Equation (115), that are present in the solution. This result is used as the basis for comparison to the numerical results calculated in **Section 4.6.2**.

Section 4.6.2 provides the results to the analysis of the nonlinear diffusion equation. [Figure 14](#) shows the theoretical and numerical calculations of the time-constants subject to the initial conditions shown in [Figure 15](#).

Section 4.6.3 relates the results calculated in Section 4.6.2 to population expansion and relates them to the paper written by Newman and Sagan (1981). Figure 18 shows the self-similarity that arises out of the solution and Figure 19 shows the position versus time to allow the reader to get a feel for the velocities of the expanding profile. The velocities are shown to decrease with increasing nonlinearity which is in good agreement to the results arrived at by Newman and Sagan (1981).

2. Background

2.1 Diffusion

Diffusion phenomena occur widely in nature. From problems in filtration, phase transition, biochemistry and dynamics of biological groups, diffusion is a very important phenomenon deserving intense scrutiny. Diffusion is very prevalent in understanding the Earth-Space environment and how it affects those on the surface. The Earth's magnetosphere is a very complex environment and one process that drives its dynamics is diffusion. One important diffusion process occurring in this environment is thermal diffusion. This is described by the partial differential equation

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x} \left(D \frac{\partial T}{\partial x} \right), \quad (2)$$

where D is the diffusion coefficient. Equation (2) comes from the basic continuity equation

$$\frac{\partial T}{\partial t} + \nabla \cdot (\vec{J}) = 0 \quad (3)$$

which is a statement of energy conservation and Fick's law

$$\vec{J} = -D\nabla T \quad (4)$$

which is a linear assumption (if D is constant) for the energy flux. This doesn't apply to all situations, but it is a simplifying assumption [Newman and Sagan, 1981]. This type of diffusion occurs when gradients in the spatial temperature profile are small, and Equation

(4) describes the fact that heat energy diffuses from regions of high temperature to regions of lower temperature. This is called linear diffusion because the unknown function T and its partial derivatives are raised to the first power and at most one of these appears in any one term of the equation. The diffusion coefficient, D , scales in the following manner

$$D \sim \frac{\lambda^2}{\tau} \quad (5)$$

where λ is the “mean-free path,” and $\tau = 1/\nu$ is the time between collisions, and ν is the collision frequency.

Often, however, the diffusion occurring in nature is nonlinear, and as is shown later the spatial-partial derivative becomes nonlinear. The time history of the temperature profile, the velocity at which the front of the profile expands, and more specifically the timescale, over which this occurs, is the focus of this thesis. Three different numerical methods will be employed to explore this equation. The three methods are an explicit method, an implicit method, and a method that is a mixture of the two known as “Crank-Nicolson”. All three of these methods have benefits and drawbacks that are useful in analyzing this equation, but ultimately the Crank-Nicolson method will be utilized.

2.2 *Nonlinear Science*

In order to better understand the processes occurring in Nature it is necessary to look at the nonlinearities existing in Nature. Several common nonlinear phenomena are weather

patterns, avalanches, fluid dynamics, forest fires, hurricanes and tornados, and global warming. These phenomena affect everyday life on the surface of the planet. Mathematical models have been developed to describe some of these systems, but unfortunately nature does not always comply with what the model will predict. The Madden-Julian Oscillation [Fell, 2005], the Climatology and Persistence model, and the Beta and Advection Model (two models used in the forecast of hurricanes [Hurricane Alley, 2005]) are some examples of mathematical models used to help predict nonlinear phenomena. Improvements need to be made to these models in order to fully describe what is occurring in nature. One phenomenon of great importance is nonlinear diffusion processes in plasmas. These processes can help explain the complex interaction of the Earth's magnetosphere with the space environment, namely, how the magnetic field interacts with plasmas [Smolyakov and Khabibrakhmanov, 1998]. Other important nonlinear processes arise in population expansion. Reduction of the amount of homozygotes in a population, the drunkard's walk, and colonial expansion in the early United States are all examples that can be explained by a diffusion process. For the latter, the diffusion coefficient should be decreased to simulate the difficulties of overland travel while it should be increased to simulate the relative ease of ocean travel to get to California [Newman and Sagan, 1981]. For population-dependent diffusion, the coefficient can be employed to represent social and territorial influences which make population expansion nonlinear.

2.3 Plasmas

Heat diffusion in plasmas is another example of an inherently nonlinear phenomenon. This is because the diffusion coefficient is dependent on the collision frequency between the charged particles. A collision between two charged particles is significantly different from a collision between two hard, billiard-ball-like spheres. The Coulomb force, which is an inverse square force (and hence long range), implies that the collision frequency decreases as the relative velocity of the particles increases. Since temperature is simply a measure of the random velocity, the collision frequency decreases as the temperature increases as well. All these facts conspire to create a diffusion coefficient (parallel to any background magnetic field) that depends on the temperature,

$$D_{\parallel} \propto T^{5/2}. \quad (6)$$

Inserting this into the diffusion equation results in a nonlinear partial differential equation that cannot be solved exactly. In addition, because any background magnetic field restricts particle motion perpendicular to that field, the diffusion coefficient in the perpendicular direction is reduced,

$$D_{\perp} \propto T^{-1/2}. \quad (7)$$

This anisotropy and nonlinearity presents a difficult theoretical problem. Clearly, this rich physical system deserves attention. This was the initial motivation for looking at a nonlinear D .

2.4 Population Expansion

Many factors contribute to whether or not a species will expand to new territory. Population growth, saturation of an environment, the carrying capacity of a planetary environment, technological advances in society, how long the society is in the expansion phase, and the sheer distance between new habitable worlds are all contributing factors to how long it will take an expanding population to reach a new equilibrium. Several examples already exist: the muskrat and the drunkard's walk (random motion) are two common examples of population expansion that can be modeled by diffusion. The above factors all contribute to the level of nonlinearity in the diffusion of a species. The goal is to determine the timescales over which this expansion occurs and how fast the population will reach the new equilibrium. Newman and Sagan (1981) give an adequate treatment of interstellar population expansion and some of their results will be reproduced by different methods throughout the course of this thesis.

In particular, the steady-state temperature profiles seen in [Figure 1](#) are used as a basis for an initial population distribution. The linear case can be solved for exactly and this solution is paramount in developing a numerical code that can properly model the nonlinearity of Equation (1). These steady-state profiles are used as a basis for the expansion of a particular population distribution. The more nonlinear the problem gets, the steeper the "front" of the distribution becomes. [Figure 2](#) shows how the steady-state profiles are going to be modeled as "expanding" out into the galaxy from some initial region. Since we are interested in a time history of the solution, [Figure 2](#) shows how the

various initial conditions now have the ability to expand out into free space. The velocity of the “front” and how long the solution takes to approach equilibrium are going to be compared to the exact solution of the linear case (i.e., how long does it take Figure 2 to look like Figure 1). Of course, some simplifying assumptions will be made to make some of the values easier to calculate. This is discussed in greater detail later in the thesis.

3. The Linear Diffusion Equation

3.1 Modeling the Linear Diffusion Equation

Before expanding the analysis to the nonlinear diffusion equation, the linear equation will be looked at first. In order to solve the linear diffusion equation it is useful to non-dimensionalize the equation. This means to scale the equation in such a way that it becomes dimensionless. This is due to the fact that the computer doesn't care what the physical units of the equation are, but the numbers it has to calculate. It also allows the solution to be expressed in terms of universal parameters.

The general diffusion equation can be solved exactly [Chen, 1984]. Starting with the general one-dimensional linear diffusion equation

$$\frac{\partial f}{\partial t} = D \frac{\partial^2 f}{\partial x^2}, \quad (8)$$

and applying Dirichlet boundary conditions, $f(0) = f(L) = 0$, the exact solution can be found using the common technique of separation of variables. Here, the solution is assumed to be of the form

$$f(x, t) = g(x)h(t). \quad (9)$$

Plugging this assumption into Equation (8) and separating variables Equation (8) takes the form

$$\frac{1}{h} \frac{dh}{dt} = -C = D \frac{1}{g} \frac{d^2 g}{dx^2}, \quad (10)$$

where C is the separation constant. The solution to the temporal equation

$$\frac{dh}{dt} = -Ch, \quad (11)$$

is an exponential

$$h(t) = e^{-Ct}, \quad (12)$$

while the general solution to the spatial portion of Equation (10)

$$\frac{d^2 g}{dx^2} = \frac{-C}{D} g \quad (13)$$

is, in the present case, most usefully expressed in terms of trigonometric functions

$$g(x) = A \sin\left(\sqrt{\frac{C}{D}}x\right) + B \cos\left(\sqrt{\frac{C}{D}}x\right). \quad (14)$$

However, the boundary condition $f(0)=0$ requires that $B = 0$. Also, the boundary $f(L)=0$ quantization condition implies

$$\sqrt{\frac{C}{D}}L = n\pi \quad \text{for } n = 1, 2, 3, \dots \quad (15)$$

The separation constant therefore takes on discrete allowable values:

$$C_n = \frac{D}{L^2} n^2 \pi^2 \quad \text{for } n = 1, 2, 3, \dots \quad (16)$$

Each mode of the exact solution decays exponentially with a time constant

$$\tau_n = \frac{1}{C_n}. \quad (17)$$

The most general solution is a linear combination of all the product solutions. In other words, it is a Fourier series.

$$f(x,t) = \sum_{n=1}^{\infty} A_n e^{-t/\tau_n} \sin\left(n\pi \frac{x}{L}\right) \quad (18)$$

with

$$\tau_n = \frac{L^2}{Dn^2\pi^2} \quad (19)$$

Note that each spatial scale, i.e., each mode n , decays on a different time scale τ_n . In particular, shorter wavelength structures (higher n) decay more quickly. Now that the exact solution to the linear diffusion equation is known the next step is to scale the equation for a numerical solution. In order to eliminate units, the solution will be expressed in terms of dimensionless variables \bar{t} and \bar{x} that are defined as follows:

$$\bar{t} = \frac{t}{T} \quad (20)$$

and

$$\bar{x} = \frac{x}{X}, \quad (21)$$

Where T and X are constants that are appropriate for the problem of interest. For example, if $T = 1s$ and $X = 1m$ then \bar{x} and \bar{t} are simply expressed in SI units. A more “physical” approach is to choose “natural” length and time scales, those that are natural for the particular system of interest.

For this analysis all spatial scales, from large to small, are of interest. So the only spatial scale is determined by the boundary conditions. Therefore, $X = L$. Since all spatial scales are of interest, from Equation (18), all temporal scales are of interest as well. Hence, the “natural” temporal scale associated with the length scale $X = L$ can be obtained by forcing the dimensionless coefficients in the differential equation to be unity.

$$\frac{1}{T} \frac{\partial f}{\partial t} = \frac{D}{X^2} \frac{\partial^2 f}{\partial x^2} \quad (22)$$

or

$$\frac{\partial f}{\partial \bar{t}} = \left(\frac{DT}{L^2} \right) \frac{\partial^2 f}{\partial \bar{x}^2} \quad (23)$$

It follows from Equation (23) that in order for the coefficient of the derivatives to be unity, the timescale must take on the value

$$T = \frac{L^2}{D} \quad (24)$$

or

$$T = \pi^2 \tau_1, \quad (25)$$

and the dimensionless linear diffusion equation becomes

$$\frac{\partial f}{\partial \bar{t}} = \frac{\partial^2 f}{\partial \bar{x}^2}. \quad (26)$$

So expressing the exact solution in terms of the new, dimensionless variables,

$$f(\bar{x}, \bar{t}) = \sum_{n=1}^{\infty} A_n e^{-n^2 \pi^2 \bar{t}} \sin(n\pi \bar{x}) \quad (27)$$

where

$$\frac{t}{\tau_n} = \frac{\bar{t}T}{\tau_n} = \frac{\bar{t}\pi^2 \tau_1}{\tau_n} = \bar{t}\pi^2 n^2 \quad (28)$$

and

$$n\pi \frac{x}{L} = n\pi \bar{x} \quad (29)$$

Hence, while Equation (18) is the solution to Equation (8) over the domain $0 < x < L$,

Equation (27) is the solution to Equation (22) over the domain $0 < \bar{x} < 1$.

Unlike most cases, the appearance of π can be eliminated simply by rescaling x and t .

Let the new scalings be

$$X = \frac{L}{\pi} \quad (30)$$

and

$$T = \tau_1 = \frac{L^2}{D\pi^2} \quad (31)$$

The differential equation is still Equation (26) because the coefficient is still unity

$$\frac{DT}{X^2} = D \frac{L^2}{D\pi^2} \frac{\pi^2}{L^2} = 1 \quad (32)$$

and the solution becomes

$$f(\bar{x}, \bar{t}) = \sum_{n=1}^{\infty} A_n e^{-n^2 \bar{t}} \sin(n\bar{x}) \quad (33)$$

But now the domain of the new solution is $0 < \bar{x} < \pi$. Either of the two scalings mentioned above are perfectly acceptable. Ultimately, the numerical solution must be expressed in terms of real, dimensional variables, and the inverse transforms $(\bar{t} \rightarrow t, \bar{x} \rightarrow x)$ need to be made.

3.2 Developing the Code

In order to address this subject to its fullest extent modeling needs to be performed. This requires a numerical code that can accurately model the general diffusion equation. The general diffusion equation is a second order partial differential equation of the parabolic

type. This means that it is first order in one of its independent variables and second order in the other; hence, it is similar in form to that of a parabola $y = x^2$. However, in order to solve the equation numerically, the continuous derivatives must be approximated by finite differences. The most straightforward way to numerically model the linear diffusion equation is to use what is commonly referred to as an explicit method. This method provides an “explicit” means to compute the spatial values for a future time based on the present values [Chapra and Canale, 2002]. This method takes the known values of the dependent variable f and uses them to predict the subsequent values in time. The explicit method uses two differencing techniques to solve the equation: a forward difference for the time variable, and a centered difference for the spatial variable. Utilizing these two differencing techniques the diffusion equation is transformed into the following form,

$$\frac{T_j^{i+1} - T_j^i}{\Delta t} = \frac{T_{j+1}^i - 2T_j^i + T_{j-1}^i}{(\Delta x)^2}. \quad (34)$$

Here $T_j^i = T(x_j, t_i)$ and

$$t_i = t_0 + i\Delta t \quad i = 1, \dots, M, \quad (35)$$

$$x_j = x_0 + j\Delta x \quad j = 1, \dots, N \quad (36)$$

respectively, and t_0 and x_0 are the initial temporal and spatial points. As was shown in the previous section, the diffusion coefficient D can be “scaled away” by a judicious choice of length and time scales. In addition, the dependent variable f has been replaced by T because this is the finite difference form of the diffusion equation and we are looking at temperature. Note that in the above form of the equation the subscript j represents the value in space and the superscript i represents the value in time. The two

terms Δt and $(\Delta x)^2$ represent the temporal and spatial resolution of the environment.

Solving this equation for T_j^{i+1} leads to the following,

$$T_j^{i+1} = T_j^i + \frac{\Delta t}{(\Delta x)^2} (T_{j+1}^i - 2T_j^i + T_{j-1}^i). \quad (37)$$

It is easy to see with this form of the equation that the future temperature values (T_j^{i+1} for all j) are calculated based on the current known temperature values (T_j^i for all j). While this is the easiest way to numerically solve the equation, the explicit method does have one drawback; it is only conditionally stable. This means that the numerical equation has to meet certain requirements in order for the solution to make sense as time goes on. Specifically, the time-step must satisfy

$$\Delta t < \frac{(\Delta x)^2}{2}. \quad (38)$$

This is known as von Neumann stability analysis (see Appendix A: Additional Derivations). This condition places limits on how finely the region of interest is resolved and how short of a time step can take place [Chapra and Canale, 2002]. This is not good enough to do a thorough analysis. It restricts the choice of time step to be small making computation time long so if the desired analysis requires a larger time step this method is not valid. So a method that is unconditionally stable needs to be developed. This is found in the implicit method. This method has a solution that is stable regardless of the time-step used in the numerical algorithm. Unfortunately, this method solves for the temperature at the next time step in terms of the values of the temperature at the next time step. In a sense it tries to solve the equation by “implying” what the solution will look like. It uses the same differencing techniques utilized by the explicit method, but instead

of using the values at the current time step, the values at the next time step are used. The finite difference equation becomes,

$$\frac{T_j^{i+1} - T_j^i}{\Delta t} = \frac{T_{j+1}^{i+1} - 2T_j^{i+1} + T_{j-1}^{i+1}}{(\Delta x)^2}. \quad (39)$$

Manipulating this equation to collect like terms puts it into the following form.

$$-\lambda T_{j+1}^{i+1} + (1 + 2\lambda)T_j^{i+1} - \lambda T_{j-1}^{i+1} = T_j^i \quad (40)$$

Note that in the above form of the equation the substitution $\lambda = \Delta t / (\Delta x)^2$ has been used.

The parameter λ shows that the values of Δx and Δt are not important independently, but only the combination that appears in λ . This is simply a reflection of the stability condition in Equation (38). This equation leads to a set of algebraic equations that needs to be solved simultaneously. It is, in effect, a matrix equation

$$[A]\{x\} = \{B\} \quad (41)$$

Where A is the tri-diagonal coefficient matrix,

$$A = \begin{bmatrix} (1 + 2\lambda) & -\lambda & 0 & 0 \\ -\lambda & (1 + 2\lambda) & -\lambda & 0 \\ 0 & -\lambda & (1 + 2\lambda) & \ddots \\ 0 & 0 & \ddots & \ddots \end{bmatrix}, \quad (42)$$

x is the function vector (T_j^{i+1} for all $j = 1, \dots, N$), and B is the current time-step vector (T_j^i for all $j = 1, \dots, N$). The matrix A is an $N \times N$ matrix, where N is the number of grid points in the spatial dimension, and $\Delta \bar{x} = 1 / (N - 1)$ is the spatial resolution. Several techniques can be utilized to solve this system of equations. Gauss elimination is one technique that can exactly solve systems of linear algebraic equations [Chapra and Canale, 2002]. Gauss elimination involves two steps: forward elimination and backward

substitution. The forward elimination portion of the technique utilizes the bulk of the computing time and does not prove to be efficient when the resolution of the spatial dimension is very high (i.e., large matrices), because Gauss elimination takes approximately $N^3/3$ computations to complete [Chapra and Canale, 2002]. In other words, the Gauss elimination method is very inefficient in general at calculating the matrix inverse because it is an exact method. However, because A is tridiagonal (most of the matrix elements are zero), exact methods can be employed efficiently.

3.3 *LU Decomposition*

A faster means to calculate multiple solutions of the matrix inverse is necessary. This leads to the LU Decomposition method which is a different means of expressing Gauss elimination. Chapra and Canale explain this method in the following way: it provides an efficient means to compute the matrix inverse. It “separates the time-consuming forward elimination of the matrix $[A]$ from the manipulations of the right-hand side $\{B\}$. Thus, once $[A]$ has been “decomposed,” multiple right-hand-side vectors can be evaluated in an efficient manner” [Chapra and Canale, 2002].

Now suppose that Equation (41) can be expressed as an upper triangular system. Here, elimination is used to reduce the system to upper triangular form.

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} \quad (43)$$

Where I have used a 3 x 3 matrix as an example. In reality, the matrix will be N x N, where N is the number of spatial grid points. Written concisely Equation (43) takes the form

$$[U]\{x\} - \{D\} = 0 \quad (44)$$

Next assume there exists a lower diagonal matrix with 1's on the diagonal.

$$[L] = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \quad (45)$$

This has the property that when left-hand multiplication is performed on Equation (44), the result is Equation (46).

$$[L][U]\{x\} - \{D\} = [A]\{x\} - \{B\} \quad (46)$$

If this equation is true, then it follows from matrix multiplication that

$$[L][U] = [A] \quad (47)$$

and

$$[L]\{D\} = \{B\} \quad (48)$$

There is a simple two-step strategy that can be used to obtain solutions utilizing LU Decomposition:

1. *LU decomposition step.* $[A]$ is factored or “decomposed” into lower $[L]$ and upper $[U]$ triangular matrices.
2. *Substitution step.* $[L]$ and $[U]$ are used to determine a solution $\{x\}$ for a right-hand side $\{B\}$. This step consists of two steps itself. First, Equation (48) is used to generate an intermediate vector $\{D\}$ by forward

substitution. Then, the result is substituted into Equation (44) which can be solved by back substitution for $\{x\}$.

In order to decompose $[A]$ it is necessary to multiply row 1 by a factor

$$f_{21} = \frac{a_{21}}{a_{11}} \quad (49)$$

and subtract the result from the second row to eliminate a_{21} . This is a specific choice for L , but not the only one (this is the Doolittle method for determining L and U). Utilizing Equation (49) matrix $[A]$ can therefore be written (after the elimination step) as

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ f_{21} & a'_{22} & a'_{23} \\ f_{31} & f_{32} & a''_{33} \end{bmatrix} \quad (50)$$

Where a' and a'' are the values of the upper triangular matrix (the “ ‘ “ indicates that an elimination was performed on the row and these are the resulting values after the respective multiplications and subtractions). From this the forward-substitution step can be written concisely as

$$d_i = d_i - \sum_{j=1}^{i-1} a_{ij} b_j \quad \text{for } i = 2, 3, \dots, n \quad (51)$$

In a similar fashion the back-substitution step can be written concisely as

$$x_n = d_n / a_{nn} \quad (52)$$

$$x_i = \frac{d_i - \sum_{j=i+1}^n a_{ij} x_j}{a_{ii}} \quad \text{for } i = n-1, n-2, \dots, 1 \quad (53)$$

This reduces the computation in the linear case, where the forward substitution is the same for each time step, but in the nonlinear case, it's different for each step.

3.4 Crank-Nicolson Method

A common method that is often used is the Crank-Nicolson method. This method is second-order accurate in both space and time [Chapra and Canale, 2002]. The increase in accuracy is due to the fact that it averages both the explicit and implicit methods; combining them into one difference equation.

$$\frac{T_j^{i+1} - T_j^i}{\Delta t} = \frac{1}{2} \left[\frac{T_{j+1}^i - 2T_j^i + T_{j-1}^i}{(\Delta x)^2} + \frac{T_{j+1}^{i+1} - 2T_j^{i+1} + T_{j-1}^{i+1}}{(\Delta x)^2} \right] \quad (54)$$

This equation can be rearranged to resemble the fully implicit method.

$$-\lambda T_{j-1}^{i+1} + 2(1 + \lambda)T_j^{i+1} - \lambda T_{j+1}^{i+1} = \lambda T_{j-1}^i + 2(1 - \lambda)T_j^i + \lambda T_{j+1}^i. \quad (55)$$

It can be seen that this is similar in form to Equation (40). All of the terms on the right-hand side are known. This makes the determination of the solution more accurate. Since the terms on the right-hand side of Equation (55) are known the same LU Decomposition algorithm can be utilized.

It is of theoretical interest to determine if the average chosen by Crank and Nicolson indeed results in the minimum error. To answer this, Equation (54) can be modified to change the ratio of implicit to explicit used in the calculation. This modified version of the difference equation introduces a parameter θ .

$$\frac{T_j^{i+1} - T_j^i}{\Delta t} = \left[(1 - \theta) \left(\frac{T_{j+1}^i - 2T_j^i + T_{j-1}^i}{(\Delta x)^2} \right) + \theta \left(\frac{T_{j+1}^{i+1} - 2T_j^{i+1} + T_{j-1}^{i+1}}{(\Delta x)^2} \right) \right] \quad (56)$$

This form is chosen because the sum of the coefficients must be unity in order to be consistent with the original differential equation. Here the parameter θ can vary between 0 and 1. When θ is 0 the method becomes fully explicit and when θ is 1 the method becomes fully implicit. It is important to note that the Crank-Nicolson method is a special case where θ is equal to $1/2$. When θ is less than $1/2$ the solution becomes unstable if the time-step does not satisfy the stability condition for the explicit method (see Equation 38); when θ is greater than $1/2$ the solution behaves more like a fully implicit method. The parameter θ can be varied to find the value that will yield the lowest error between the numerical and exact solutions. Since the exact solution is known, a good “figure of merit” is the Absolute Root Mean Squared (RMS) error.

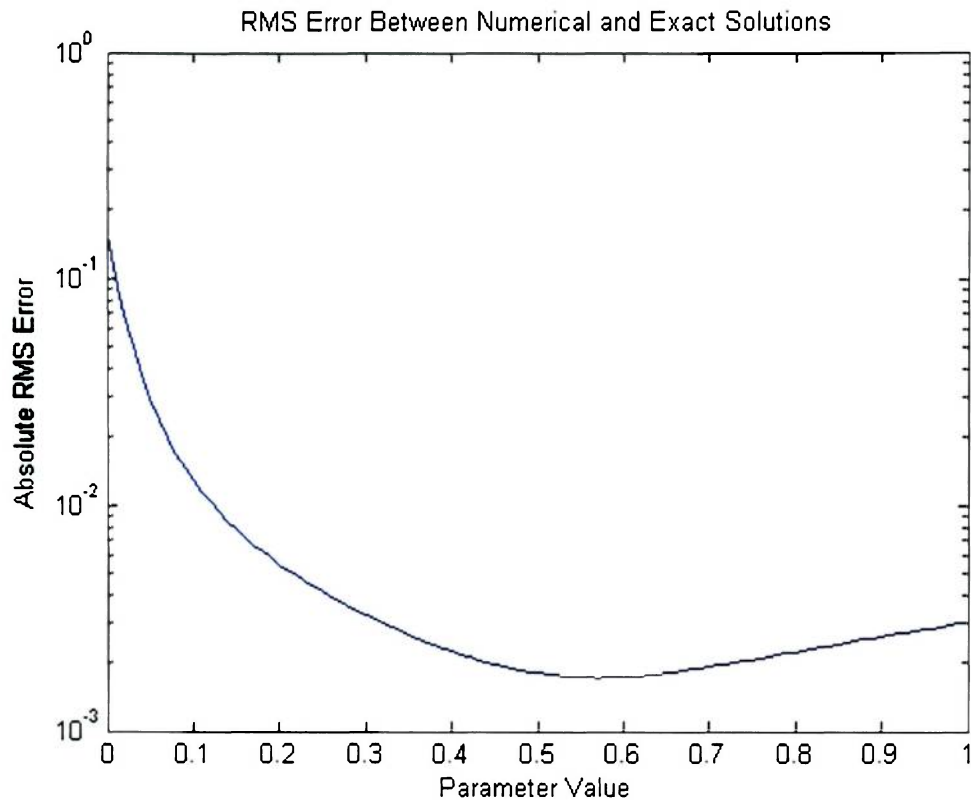


Figure 3. Value of Parameter that Yields Lowest Absolute RMS Error.

In [Figure 3](#) the lowest Absolute RMS error occurs when the parameter θ is equal to 0.57. The exact case used to create this graph is shown in the appendix. It is important to note that as the ratio of $\Delta t/(\Delta x)^2$ increases, the value of θ that yields the lowest RMS error also increases, making the method more implicit (See Appendix B: Code VaryingParameter.m). This means that the above figure will change according to the different initial conditions that it is given. This is due to the nature of the explicit method and its stability condition.

It turns out that for the case of interest in this thesis, the total RMS error is only weakly dependent on the value of θ . For example, in one case discussed later, the specific

conditions are $\Delta x = 1/300$ and $\Delta t \approx 1 \cdot 10^{-3}$. With these particular values the minimum error for the Crank-Nicolson method is within one order of magnitude from the errors for fully explicit and fully implicit, Table 1 (i.e., $\theta = 0,1$). For this reason, it is not critical which value of θ is used. However, for reasons of stability, we choose the Crank-Nicolson method.

Table 1. Comparison of Lowest Absolute RMS Error for Crank-Nicolson Method.

Parameter (θ) Value	Lowest Absolute RMS Error	Δt value
0.01	0.9186	0.0005
0.01	0.8469	0.0006
0.08	0.7801	0.0007
0.13	0.7176	0.0008
0.17	0.6591	0.0009
0.20	0.6042	0.0010
0.22	0.5527	0.0011
0.24	0.5044	0.0012
0.26	0.4588	0.0013
0.28	0.4159	0.0014
0.29	0.3754	0.0015
0.30	0.3371	0.0016
0.31	0.3008	0.0017
0.32	0.2665	0.0018
0.33	0.2340	0.0019
0.34	0.2031	0.0020
0.34	0.1736	0.0021
0.35	0.1457	0.0022
0.35	0.1190	0.0023
0.36	0.0935	0.0024
0.37	0.0692	0.0025
0.37	0.0459	0.0026
0.37	0.0237	0.0027
0.38	0.0024	0.0028
1.00	0.0139	0.0029
1.00	0.0334	0.0030

Table 1 shows how the minimum RMS error of the Crank-Nicolson method varies with various values of Δt . The first column is the parameter value that yields the lowest RMS error (middle column) for the given Δt (last column). It is important to note that since the difference in the error from lowest value to highest value (when comparing the modified Crank-Nicolson to the fully implicit method) only differs by an order of magnitude, unless Δt falls into a specific range that is very small, it does not matter which method of analysis is used. Therefore, certain parts of the analysis were done utilizing the fully implicit method while others were done utilizing the Crank-Nicolson method.

3.5 Comparison of the Numerical Results to the Analytic Result

Now that the exact solution is known the next step is to solve the linear diffusion equation (Equation 8) numerically and compare to the exact solution. Equation (8) is written in such a way as to resemble Equation (41). It can now be combined with the LU Decomposition algorithm and solved. It is important to note that Equation (8) does not produce a completely filled coefficient matrix when expanded. It produces a special matrix known as a tri-diagonal matrix. This type of matrix only has the main diagonal plus the two adjacent diagonals filled with non-zero terms reducing the number of operations significantly (from N^3 to N^2 operations [Chapra and Canale, 2002]).

The algorithm is well-known and standard—it only uses the three diagonals and ignores the other elements (because they are zero). For the linear diffusion equation the constant vector on the right-hand side of Equation (41) is the current known temperatures of the field at time t_i . The upper and lower diagonals of the coefficient matrix are $-\lambda$ and the main diagonal is $\alpha = 1 + 2\lambda$. This can easily be seen when looking at Equation (40). The rest of the development of the algorithm can be seen in Appendix B: Code Gooddiffusion.m.

In order to verify that the algorithm is solving the diffusion equation correctly an initial waveform has to be given in order to view the diffusion taking place. The first initial condition chosen is a linear combination of modes with equal amplitudes,

$$T(\bar{x}, \bar{t} = 0) = \sin(\pi\bar{x}) + \sin(2\pi\bar{x}) + \dots + \sin(10\pi\bar{x}) \quad (57)$$

The initial wave form is shown in [Figure 4](#).

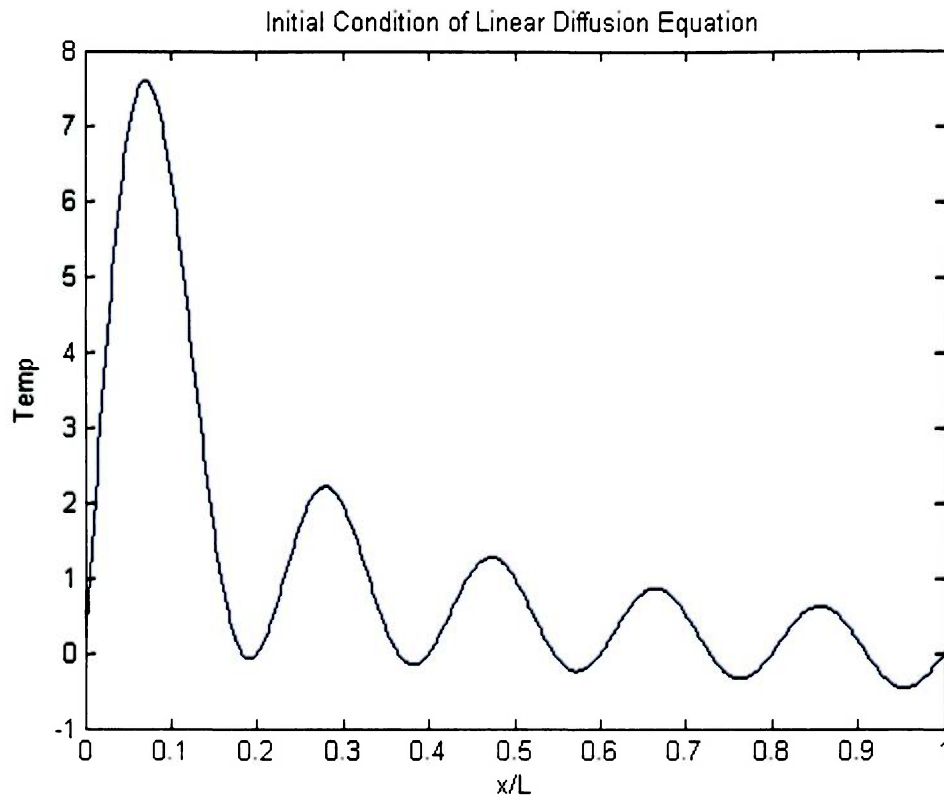


Figure 4. Initial Waveform used to Verify Code.

Note that the scales are unitless and the boundary conditions chosen are Dirichlet. Here a negative *Temp* is not relevant for the diffusion equation—only in the physical case. Three main ideas must be verified before expanding the code to handle the nonlinear case. First, the exponential decay of the mode amplitudes with τ_n as derived has to be shown; second, the initial waveform has to decay to zero; and third, the numerical solution should closely match the exact solution in both mode amplitude decay and waveform decay.

For the following analysis, a fully implicit method with a $\bar{\Delta x} = 1/500$, and a $\bar{\Delta t} = 0.0001$ was used. The results are shown in [Figure 5](#).

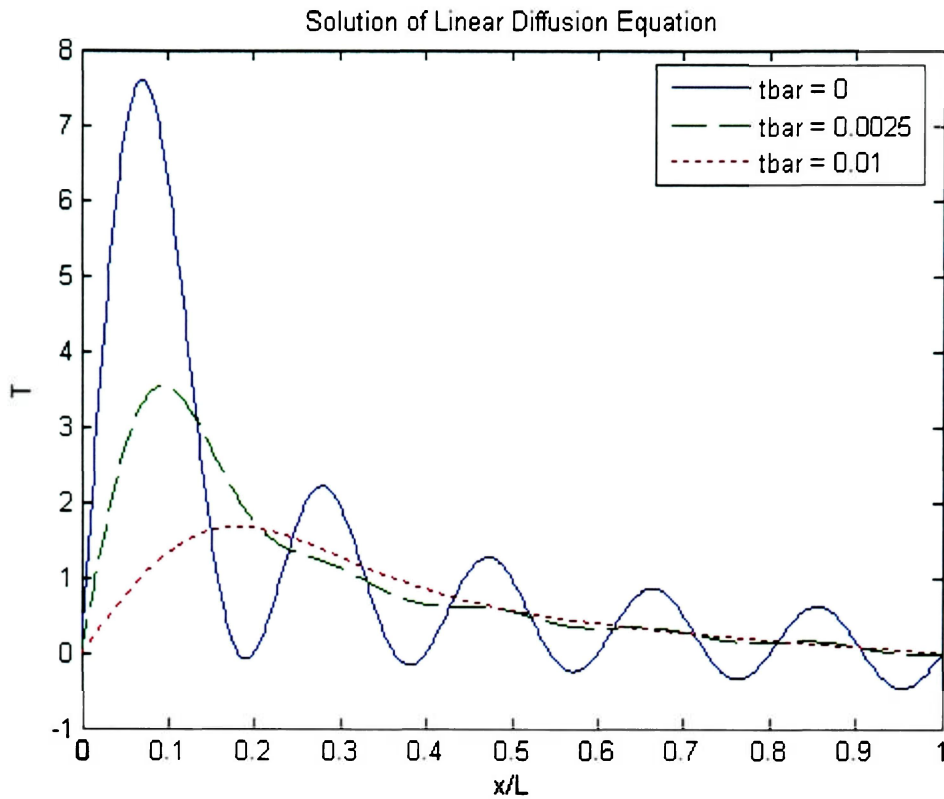


Figure 5. Time Lapse of Initial Waveform.

Figure 5 verifies the numerical code. It shows the initial waveform decaying away to zero. It also shows the higher order modes decaying away faster than the lower order modes as expected. This is due to the fact that there are Dirichlet boundary conditions and no source term present. This means that all the energy in the system should flow out of the boundaries (at a rate proportional to the slope at the boundaries).

The next idea that needs to be verified is that the mode amplitudes tend to zero over time. The initial waveform has ten modes that comprise its shape. Each mode decays at a different rate. According to Equation (27) the higher order modes should decay faster than the lower order modes. A discrete Fourier transform was performed on the

numerical solution in order to extract the amplitudes of the different modes. This is a very common technique [Press, et al, 1987]. Figure 6 shows a time-history of the mode amplitudes.

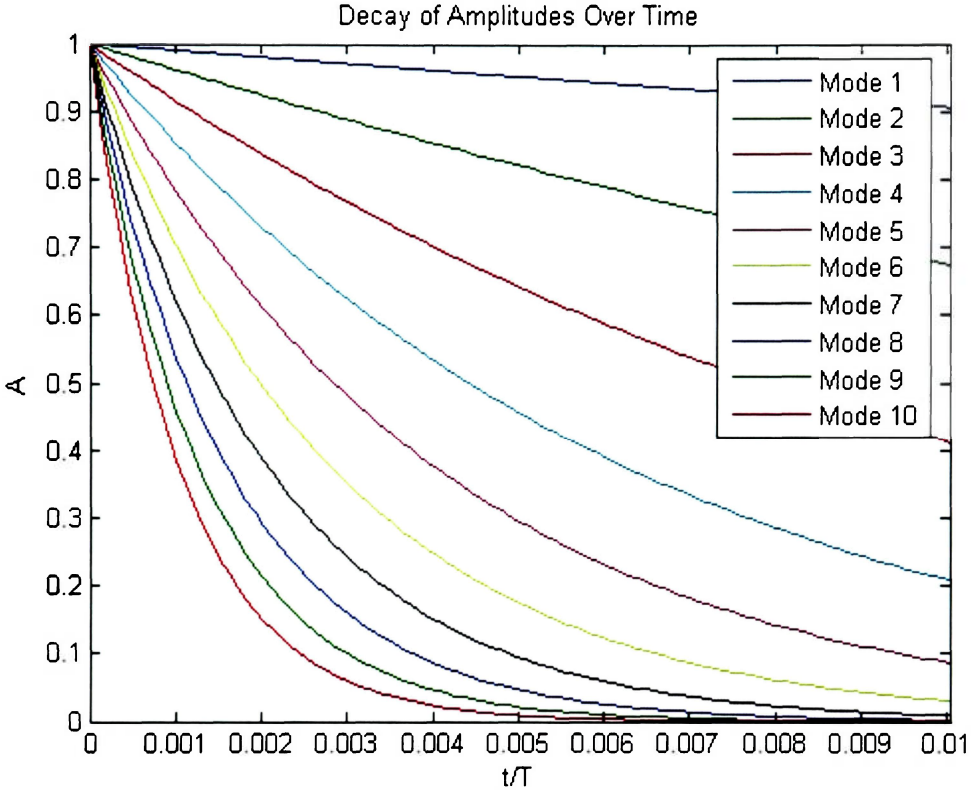


Figure 6. Decay of Amplitudes of Different Modes of Initial Waveform.

The exponential decay is very prevalent in Figure 6 with the higher order modes decaying faster than the lower order modes. These decay rates match the analytic solution (Equation 27).

Finally, the comparison to the exact solution needs to be made. Since the initial waveform is a sine series the exact solution will take the same form. This makes

comparison of the two easier. The following figures show the results of the comparison of the numerical and exact solutions.

Figure 7 shows the absolute error (exact minus numerical) between the two waveforms with respect to position.

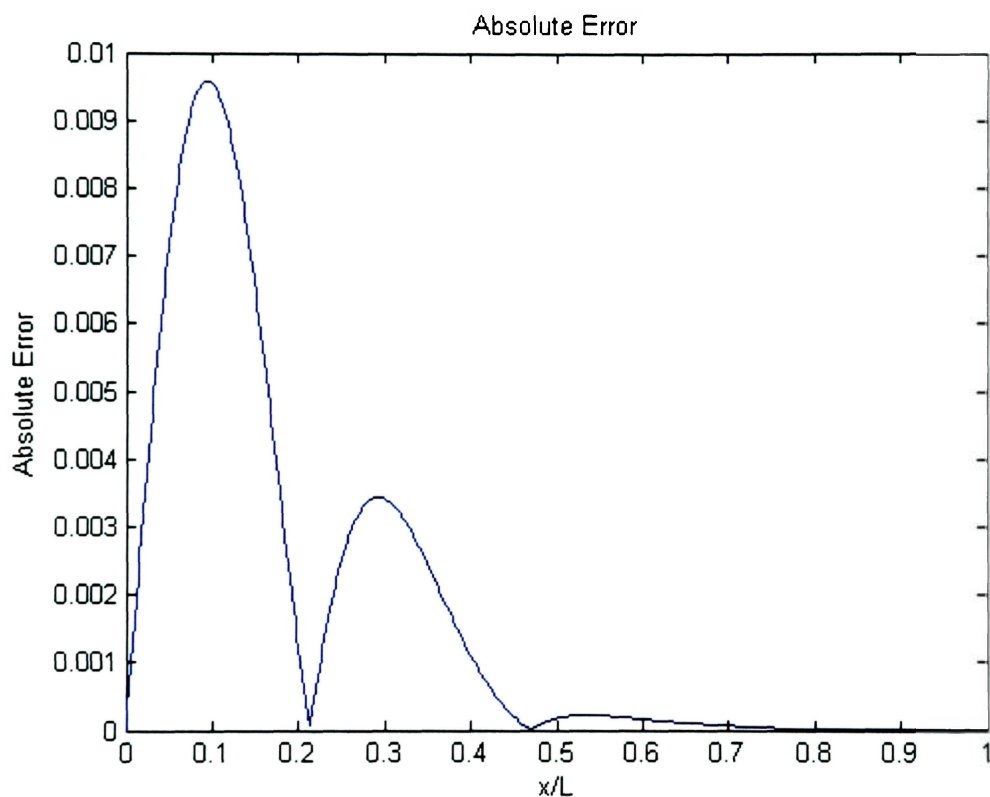


Figure 7. Absolute Error Relative to Position.

Note that Figure 7 shows the absolute error at $\bar{t} = 0.01$. It can be seen from Figure 7 that the maximum error is less than 1% (since the value of T is on the order of 1, the absolute error is approximately equal to the relative error).

The error as a function of time is also of interest. In order to incorporate all spatial locations, the RMS (root-mean-squared) error (summed over all spatial locations) is calculated at each time step, and shown in [Figure 8](#).

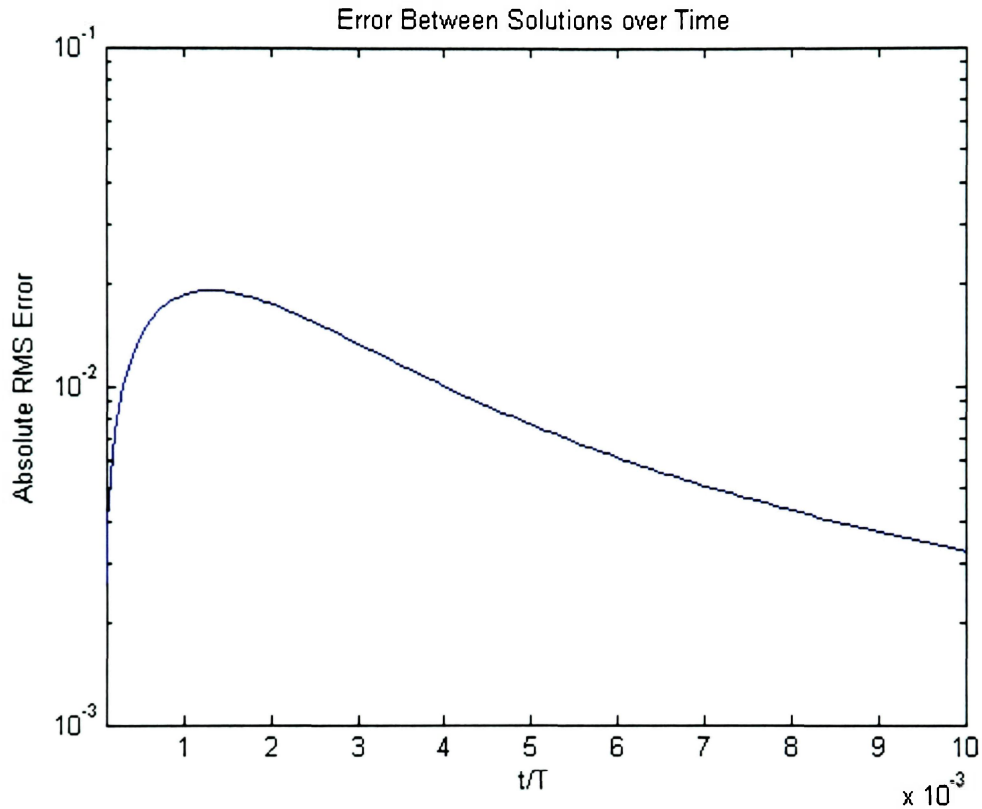


Figure 8. RMS Error Comparison between Numerical and Exact Solutions.

[Figure 8](#) shows that the two solutions are in very good agreement at the start of the solution. Unfortunately, this has no meaning because the initial conditions chosen are, by default, identical. Then the error peaks, typical of exponential processes, and slowly decays away. Eventually the error will go below machine precision due to the nature that the solutions will decay to zero. The peak in the error is on the order of 10^{-2} . Again, since the temperature is on the order of one, this absolute error is approximately equivalent to relative error.

The amplitudes of the different orders need to be compared as well. [Figure 6](#) shows the different modes of the numerical solution decaying exponentially with time. Comparing these amplitudes to the analytic prediction will reveal how closely the code solves the linear diffusion equation.

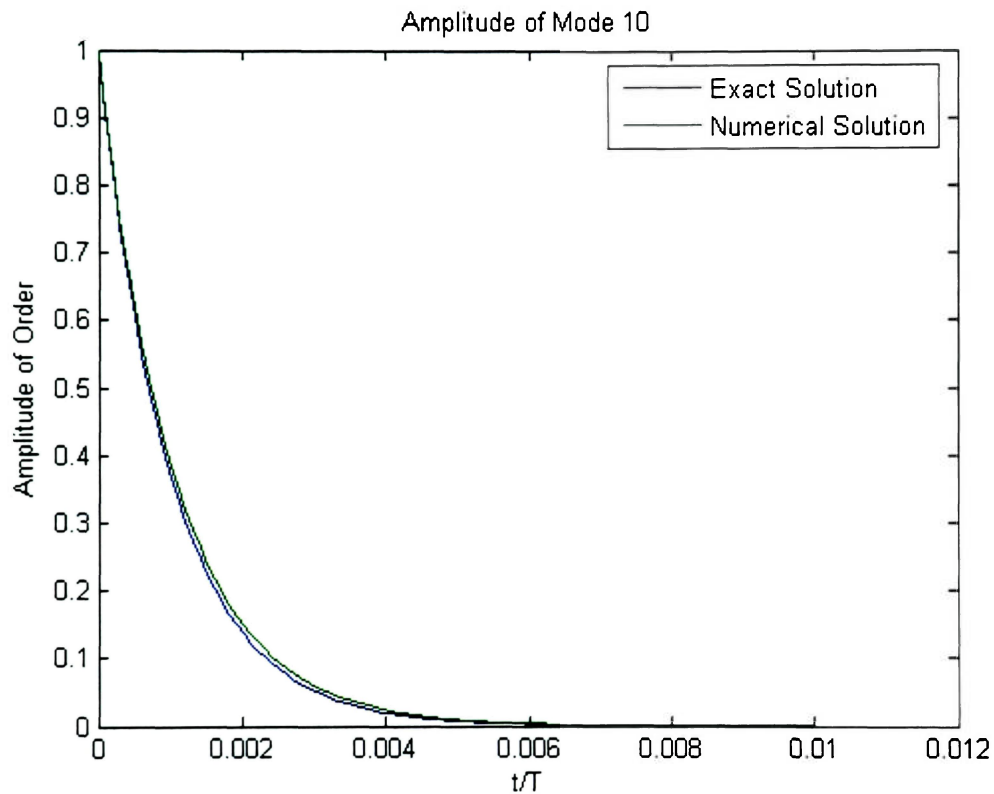


Figure 9. Comparison of Amplitudes of Selected Order Mode.

[Figure 9](#) shows the decay of the amplitudes of both the numerical and exact solutions of mode 10. There is a clear discrepancy between the two amplitudes. [Figure 10](#) shows the absolute error between the two modes.

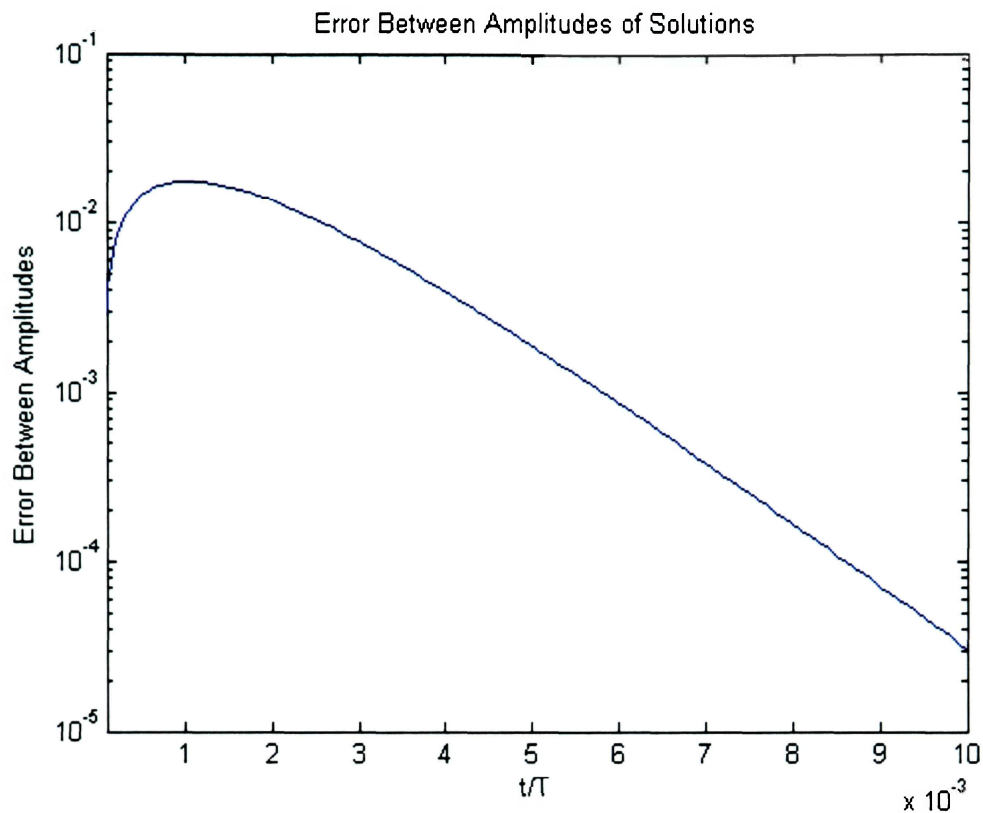


Figure 10. Absolute Error between Amplitudes of Numerical and Exact Solutions.

Here again, as in [Figure 8](#), the two amplitudes show good agreement at the start of the solution, but as time goes on the error peaks and then fades away eventually to zero. In addition, the peak in the error is of the order 10^{-2} which is similar to the RMS error between the two waveforms. It is important to note that the graphs for the other modes are similar to both [Figure 9](#) and [Figure 10](#) with the peak error being of the same order. While the maximum error is pretty high, which can be decreased by decreasing $\Delta \bar{t}$, its decay shows that the two solutions will agree over time.

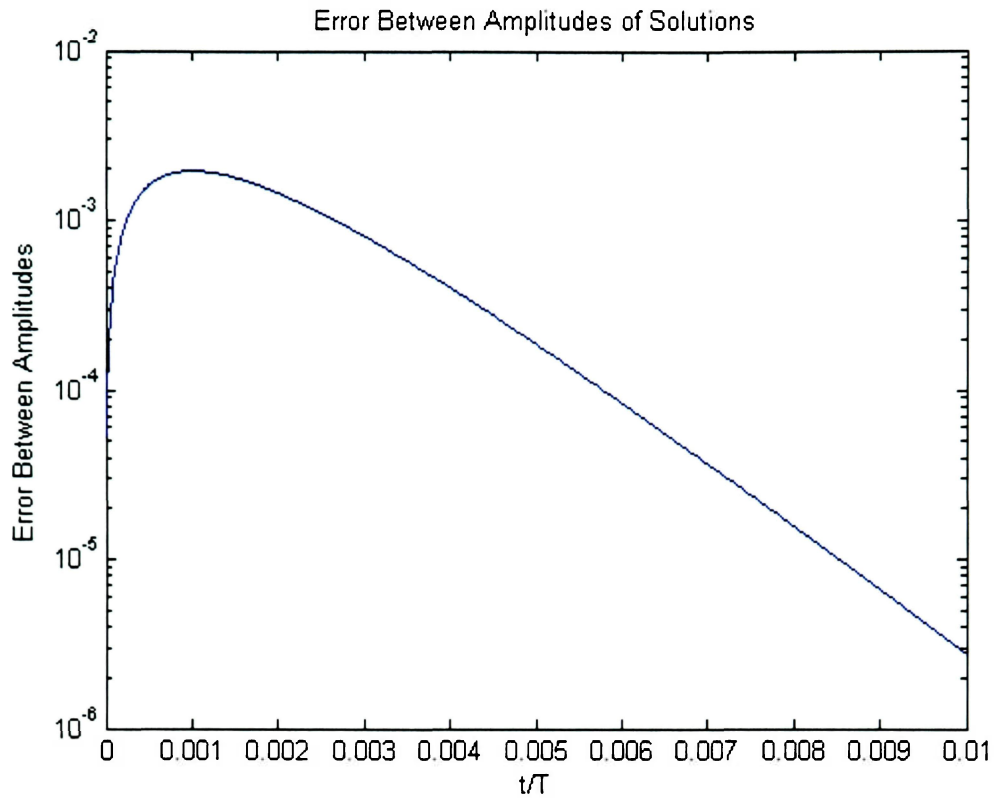


Figure 11. Absolute Error between Amplitudes of Numerical and Exact Solutions for Smaller Time-step.

Figure 11 shows how the error decreases with decreasing $\Delta \bar{t}$. The same simulation is run as above with the only change being $\Delta \bar{t} = 0.00001$ (a factor of 10 smaller). With this change the maximum error decreases by almost an order of magnitude to 10^{-3} . Since the maximum error is shown to decrease with decreasing $\Delta \bar{t}$ (comparison of Figure 10 and Figure 11) the code is then expanded to handle the nonlinear diffusion equation.

4. The Nonlinear Diffusion Equation

4.1 Development of the Nonlinear Diffusion Equation

The linear diffusion equation (Equation 8) was useful in verifying that the algorithm developed was stable. Since we are interested in nonlinear phenomena such as thermal diffusion in plasmas, and population expansion, the equation of real interest is the nonlinear diffusion equation,

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x} \left(D(T) \frac{\partial T}{\partial x} \right), \quad (58)$$

where the diffusion coefficient is a function of temperature. In particular, the case of interest is a power-law dependence.

$$D = \delta T^n \quad (59)$$

This goes along with how thermal diffusion occurs in plasmas (Equation (6) and Equation (7)). This is also the chosen diffusion coefficient by Newman and Sagan (1981). Here the diffusion coefficient is no longer a constant and depends on the temperature of the distribution. Note that δ is an arbitrary constant not to be confused with the Dirac delta function. In order for the equation to be dimensionally correct $[D] = \text{Length}^2 / \text{Time}$ and therefore δ must have the units $[\delta] = \text{Length}^2 / (\text{Time} * \text{Temperature}^n)$. It is useful to rewrite Equation (58) in the form

$$\frac{\partial T}{\partial t} = \frac{\delta}{(n+1)} \frac{\partial^2 T^{n+1}}{\partial x^2}. \quad (60)$$

When $n=0$, Equation (60) reverts to the well-known linear diffusion equation discussed above. Again, it is useful to nondimensionalize this equation. The following scaling factors will be utilized:

$$t = \frac{t}{\tau}, \quad \bar{x} = \frac{x}{L}, \quad \bar{T} = \frac{T}{T_0} \quad (61)$$

Note that because of the nonlinearity of T , it is necessary to scale the (dependent variable) temperature, in addition to the independent variables. This will be chosen for ease of analysis. After plugging in the scaling factors, Equation (60) becomes

$$\frac{\partial \bar{T}}{\partial \bar{t}} = \left[\frac{\delta \tau T_0^n}{L^2 (n+1)} \right] \frac{\partial^2 \bar{T}^{n+1}}{\partial \bar{x}^2}. \quad (62)$$

If the quantity in brackets is chosen to be unity, then in addition to arbitrarily scaling the size of the volume of interest to L (as before), this choice automatically determines the time scale $\tau = (L^2 (n+1)) / (\delta T_0^n)$. It is important to note that the coefficient on the right hand side of Equation (62) is defined in the code prior to the LU decomposition algorithm and is therefore omitted from the finite difference equation (See Appendix B: Code GoodTimeNonlinearDiffusion.m). The same forward and centered differencing techniques used on Equation (8) (that resulted in the implicit method) will be applied to Equation (62)

$$\frac{\bar{T}'_{j+1} - \bar{T}'_j}{\Delta \bar{t}} = \frac{\left(\bar{T}'_{j+1}\right)^{n+1} - 2\left(\bar{T}'_j\right)^{n+1} + \left(\bar{T}'_{j-1}\right)^{n+1}}{\left(\Delta \bar{x}\right)^2}. \quad (63)$$

Rearranging terms to place all the terms evaluated at $t = t_{i+1}$ on the left, and those at $t = t_i$ on the right

$$\bar{T}_j^{i+1} - \lambda \left[\left(\bar{T}_{j+1}^{i+1} \right)^{n+1} - 2 \left(\bar{T}_j^{i+1} \right)^{n+1} + \left(\bar{T}_{j-1}^{i+1} \right)^{n+1} \right] = \bar{T}_j^i. \quad (64)$$

Here the same substitution $\left(\lambda = \Delta t / (\Delta x)^2 \right)$ has been made. In this (nonlinear) form, it is not possible to use neither LU decomposition nor any (linear) method such as Gaussian elimination. A technique known as time linearization will be utilized to transform Equation (64) into a linear equation that is amenable to LU decomposition. Time linearization is a technique that employs a Taylor expansion of the dependent variable T with respect to time; all values of T at different spatial locations are Taylor expanded in the same manner.

4.2 Time Linearization

Time linearization, when applied to Equation (64), makes the following assumptions:

$$\bar{T}_j^{i+1} = \bar{T}_j^i + \Delta \bar{T}_j, \quad \text{for all } j \quad (65)$$

where

$$\Delta \bar{T}_j \ll \bar{T}_j^i \quad (66)$$

Physically, this means that over the interval of one time step, the temperature at each spatial location changes by a small amount, and only the first order corrections to the temperature are kept. In practice, this requires that the time step chosen be sufficiently

small (i.e. $\Delta t \approx 10^{-6}$). This ensures that both Equations (38) and (66) are satisfied. The higher order terms in Equation (64) then become

$$\left(\frac{\bar{T}'_{j+1}}{\bar{T}'_j}\right)^{n+1} \approx \left(\frac{\bar{T}'_j}{\bar{T}'_j}\right)^{n+1} \left(1 + (n+1) \frac{\Delta \bar{T}_j}{\bar{T}'_j}\right) = \left(\frac{\bar{T}'_j}{\bar{T}'_j}\right)^{n+1} + (n+1) \left(\frac{\bar{T}'_j}{\bar{T}'_j}\right)^n \Delta \bar{T}_j, \quad \text{for all } j \quad (67)$$

Plugging Equation (67) back into Equation (64) and rearranging so that it can be viewed as a matrix equation for $\Delta \bar{T}_j$, the final form of the finite difference equation becomes

$$\begin{aligned} -\lambda(n+1) \left(\frac{\bar{T}'_{j+1}}{\bar{T}'_j}\right)^n \Delta \bar{T}_{j+1} + \left(1 + 2\lambda(n+1) \left(\frac{\bar{T}'_j}{\bar{T}'_j}\right)^n\right) \Delta \bar{T}_j - \lambda(n+1) \left(\frac{\bar{T}'_{j-1}}{\bar{T}'_j}\right)^n \Delta \bar{T}_{j-1} \\ = \lambda \left(\frac{\bar{T}'_{j+1}}{\bar{T}'_j}\right)^{n+1} - 2\lambda \left(\frac{\bar{T}'_j}{\bar{T}'_j}\right)^{n+1} + \lambda \left(\frac{\bar{T}'_{j-1}}{\bar{T}'_j}\right)^{n+1} \end{aligned} \quad (68)$$

Equation (68) can be written more concisely as

$$c_1 \Delta \bar{T}_{j+1} + c_2 \Delta \bar{T}_j + c_3 \Delta \bar{T}_{j-1} = b \quad (69)$$

where

$$c_1 = -\lambda(n+1) \left(\frac{\bar{T}'_{j+1}}{\bar{T}'_j}\right)^n \quad (70)$$

$$c_2 = 1 + 2\lambda(n+1) \left(\frac{\bar{T}'_j}{\bar{T}'_j}\right)^n \quad (71)$$

$$c_3 = -\lambda(n+1) \left(\frac{\bar{T}'_{j-1}}{\bar{T}'_j}\right)^n \quad (72)$$

$$b = \lambda \left(\frac{\bar{T}'_{j+1}}{\bar{T}'_j}\right)^{n+1} - 2\lambda \left(\frac{\bar{T}'_j}{\bar{T}'_j}\right)^{n+1} + \lambda \left(\frac{\bar{T}'_{j-1}}{\bar{T}'_j}\right)^{n+1} \quad (73)$$

are all constants that must be reevaluated at every successive time step (See Appendix B: Code GoodTimeNonlinearDiffusion.m). It is clear now that Equation (68) is similar in form to Equation (40) and can utilize the LU Decomposition algorithm developed. The power of LU decomposition method is that the Gaussian elimination portion must only be performed once, and only the back substitution portion needs to be done each time step, thus reducing the number of computations by half. Unfortunately, because the

coefficients c_1 , c_2 , and c_3 now depend on the values of T at time $t = t_i$, both halves of the algorithm must be performed at each step. However, the fact that the matrix is still tridiagonal means that the method is still efficient, and hence will continue to be used. In practice, an extra step is needed to update the new temperatures using Equation (65). The algorithm then updates the constants and continues on with the solution. What is actually solved for with the nonlinear case is the change in temperature between each time interval, not the actual temperatures.

It is important to note that the Crank-Nicolson method is expanded in the same fashion. The only difference being the right hand side of Equation (73) and the addition of terms that constitute the explicit portion of the Crank-Nicolson method.

4.3 Steady-State

Now that Equation (58) can actually be solved numerically it is important to know what the steady-state solution would look like. Since one of the phenomena we are interested in is population expansion it is important to know how a given “population distribution” will look like after an advancement in technology (so to speak) allows them to travel into the galaxy away from their “home base”, or what does the temperature profile look like in a plasma or a bar for which the diffusion coefficient is not constant. It is important to note that the steady-state solution is what the profile would look like given a constant source term in the environment. What this means is that by setting the left boundary to a specific temperature it is implying that there is a source of energy, or population growth

that feeds the distribution. Another means of providing this source is to actually put a source term into the equation (see Appendix A: Additional Derivations Adding a Source Term). Here the temporal derivative is set to zero (definition of steady-state) and Equation (60) becomes

$$\frac{\delta}{(n+1)} \frac{d^2 T^{n+1}}{dx^2} = 0 \quad (74)$$

The solution for the quantity T^{n+1} must, of course, be a straight line (as long as $n \neq -1$, which will not be considered). So the steady-state solution assumes the following form.

$$T(x) = (Ax + B)^{1/(n+1)}, \quad (75)$$

where A and B are two arbitrary constants of integration and are determined by the boundary conditions. Let $T(0) = T_o = 1$ be some initial temperature at the left boundary, then

$$T(0) = T_o = (B)^{1/(n+1)} \quad (76)$$

\therefore

$$B = T_o^{n+1} \quad (77)$$

Now let

$$T(L) = T_L = (AL + T_o^{n+1})^{1/(n+1)} \quad (78)$$

Rearranging terms to solve for A

$$A = \frac{T_L^{n+1} - T_o^{n+1}}{L} \quad (79)$$

and Equation (78) becomes

$$T(x) = \left(\frac{1}{L} (T_L^{n+1} - T_o^{n+1})x + T_o^{n+1} \right)^{1/(n+1)} \quad (80)$$

Now given boundary temperature values, the steady-state temperature distribution can be calculated using this equation. What this implies is that the higher temperature boundary is acting like a source and the lower temperature boundary is acting like a sink. [Figure 12](#) shows the various steady-state distributions.

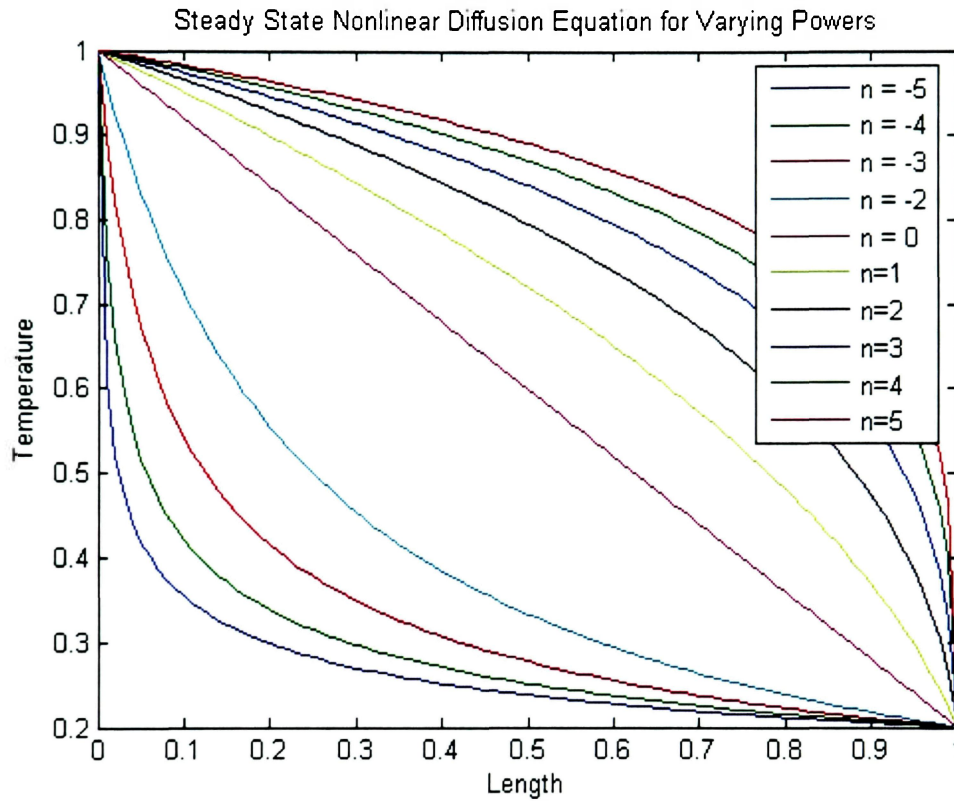


Figure 12. Steady-State Solutions for Various Powers of Nonlinearity.

Note that the solutions for negative n are concave up and the positive n are concave down. This can be explained by looking at Equation (59). For the linear case ($n=0$) the diffusion coefficient is essentially 1. When n takes on negative values it turns into a fraction of 1 (similar to $y=1/x$) and vice versa for positive values it becomes greater than 1 (depending on the temperature of course). One important note is that for the

linear case ($n=0$) the solution is a straight line as expected. Also, the case $n=-1$ is undefined.

4.4 Specific Condition of Interest

Now that the code is working properly, it will be expanded to analyze a certain condition. The interest is to see how a profile expands out into space given an initial distribution. This is important because we are interested in how a species populates the galaxy. In particular how fast the population reaches a steady-state and the timescale at which the population approaches the respective steady-state solution. Since the steady-state of the diffusion equation is known the goal is to then see how this steady-state solution will expand out into space. More specifically, the rate at which the solution expands out into space in order to reach a new equilibrium is of great importance. The distribution of interest is made up of two parts. First, in one region of space ($\bar{x} < 1/2$) the initial condition is determined by Equation (80) where $T_L = 0$ and $T_o = 1$, and in the other region of space ($\bar{x} > 1/2$) the solution is set to zero.

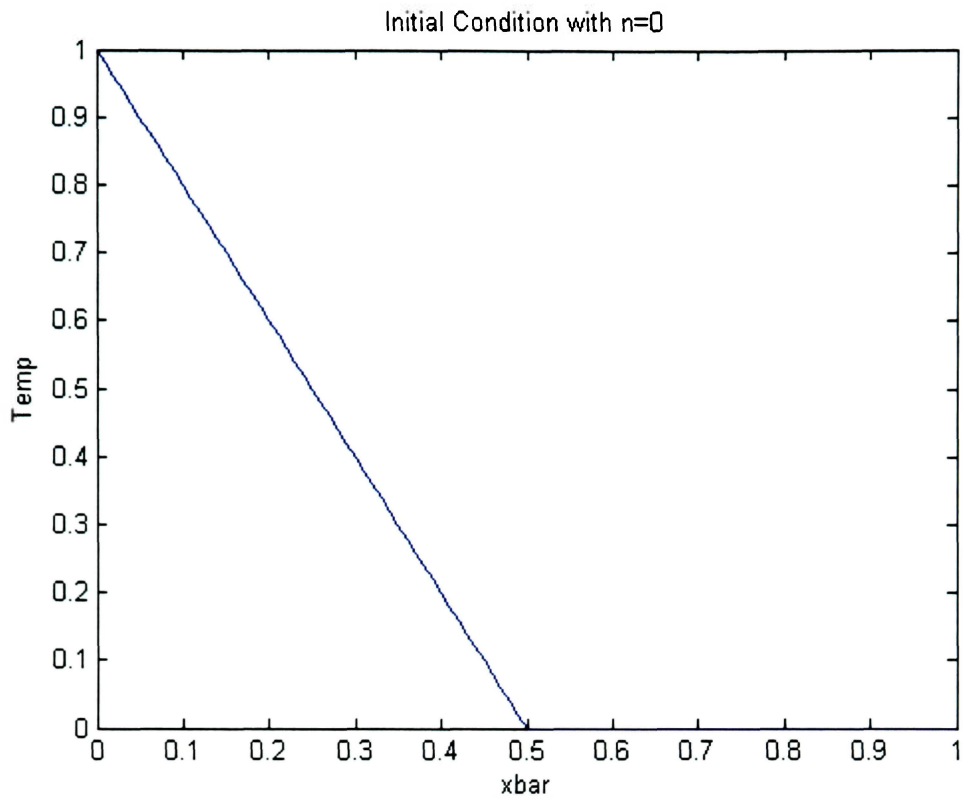


Figure 13. Initial Condition with $n = 0$.

Figure 13 shows a sample of the initial condition. Note that the initial condition for $\bar{x} < 1/2$ will change with increasing values of n .

4.5 Linear Regime of Nonlinear Equation

4.5.1 Steady-State of Selected Initial Condition

The solution to the initial condition mentioned above ([Figure 13](#)) can be solved for analytically. Beginning with the linear diffusion equation

$$\frac{\partial u}{\partial t} = c^2 \frac{\partial^2 u}{\partial x^2} \quad (81)$$

subjected to the following boundary conditions,

$$u(0, t) = T_{left} \quad (82)$$

which is the temperature on the left boundary,

$$u(L, t) = T_{right} \quad (83)$$

is the temperature on the right boundary, and

$$u(x, 0) = f(x) \quad (84)$$

is the initial condition given by the following piecewise equation to describe the region of interest

$$f(x) = \begin{cases} \left(\frac{T_{right} - T_{left}}{L/2} \right) x + T_{left} & 0 < x < L/2 \\ T_{right} & L/2 < x < L \end{cases} \quad (85)$$

The solution of which is given by the following

$$u(x, t) = u_1(x) + u_2(x, t) \quad (86)$$

where

$$u_1(x) = \left(\frac{T_{right} - T_{left}}{L} \right) x + T_{left} \quad (87)$$

is the known steady-state solution of the linear diffusion equation with fixed boundary temperatures and

$$u_2(x, t) = \sum b_n e^{-\lambda_n^2 t} \sin\left(\frac{n\pi}{L} x\right) \quad (88)$$

is the solution arrived at from separation of variables with

$$\lambda_n = \frac{cn\pi}{L} \quad (89)$$

and

$$b_n = \frac{2}{L} \int_0^L (f(x) - u_1(x)) \sin\left(\frac{n\pi}{L}x\right) dx \quad (90)$$

is used to solve for the coefficients. The only thing left to do is solve for b_n ; which after splitting up the integral and solving looks like this

$$b_n = 4(T_{right} - T_{left}) \left(\frac{\sin\left(\frac{n\pi}{2}\right)}{n^2\pi^2} \right) \quad n = odd \quad (91)$$

So the exact solution takes the following form

$$u(x,t) = \left(\frac{T_{right} - T_{left}}{L} \right) x + T_{left} + \sum_{n=odd}^{\infty} (b_n e^{-\lambda_n^2 t}) \sin\left(\frac{n\pi}{L}x\right) \quad (92)$$

It is important to note that $u_1(x)$ is the steady-state solution to the given problem. Since this is subtracted out of the general solution, the boundary temperatures are zero which means that only the odd sine modes exist. This is true only for the linear diffusion equation.

4.6 Nonlinear Regime

4.6.1 Derivation of Theoretical Timescale of Nonlinear Equation

How fast does the temperature approach the steady-state solution? We can answer this question in the limiting case where the temperature T is infinitesimally close to the

steady-state condition. Analytically, we start with some initial temperature profile (that will later be chosen to be near the steady-state profile). The short time behavior can be obtained by assuming that the temperature changes very little from this initial condition $T = T_m + T_1$ (similar to the time-linearization procedure developed above), and in this regime the equation for T_1 is simply the linear diffusion equation. Starting with the nonlinear diffusion equation

$$\frac{\partial T}{\partial t} = \beta \frac{\partial^2 T^m}{\partial x^2} \quad (93)$$

where

$$m = n + 1 \quad (94)$$

$$\beta = \frac{\delta}{m} D. \quad (95)$$

We make these replacements to reduce notational burden. Let T expand in the following manner

$$T(x, t) = T_m + T_1 \approx T(x, t_0) + T_1(x, t) \quad (96)$$

Where T_m is some “initial” state and T_1 is the difference between the current state and “initial” state. All the time dependence is in T_1 . The left side of Equation (93) becomes

$$\frac{\partial T}{\partial t} = \frac{\partial T_m}{\partial t} + \frac{\partial T_1}{\partial t}, \quad (97)$$

where $\partial T_m / \partial t = 0$ since the “initial” state is independent of time. The right hand side of Equation (93) becomes

$$\frac{\partial^2 T^m}{\partial x^2} \cong \frac{\partial^2}{\partial x^2} (T_m^m + m T_m^{m-1} T_1) = \frac{\partial^2 T_m^m}{\partial x^2} + m \frac{\partial^2}{\partial x^2} (T_m^{m-1} T_1), \quad (98)$$

where we have used

$$T^m = (T_m + T_1)^m = T_m^m + mT_m^{m-1}T_1. \quad (99)$$

Since the quantity $T_m^{m-1}T_1$ appears, it is useful to rewrite the time derivative

$$\frac{\partial T_1}{\partial t} = \frac{1}{T_m^{m-1}} \frac{\partial}{\partial t} (T_m^{m-1}T_1). \quad (100)$$

After these approximations, Equation (93) becomes

$$\frac{\partial}{\partial t} (T_m^{m-1}T_1) = \{m\beta T_m^{m-1}\} \frac{\partial^2}{\partial x^2} (T_m^{m-1}T_1) + \beta T_m^{m-1} \frac{\partial^2 T_m^m}{\partial x^2}, \quad (101)$$

which is a linear diffusion equation for the function $T_m^{m-1}(x)T_1(x, t)$ with an effective diffusion coefficient $m\beta T_m^{m-1}$ and a constant term—that acts like a source. If T_m is close to the steady-state profile T_{ss} , then

$$\frac{\partial^2}{\partial x^2} T_m^{m-1} = 0, \quad (102)$$

as concluded in Equation (74), and the source term in Equation (101) is zero. That is, our “initial condition” is almost steady-state, and this linear approximation analyzes the approach to equilibrium of the nonlinear system. Equation (101) is a linear diffusion equation with a spatially dependent diffusion coefficient. However, if we make the additional approximation that $T(x=0) \gtrsim T(L)$, then the spatial dependence of this coefficient is weak, and we can use the tools developed above for analysis. Scaling Equation (93) by using scalings similar to Equation (61)

$$\bar{t} = \frac{t}{T}, \quad \bar{x} = \frac{x}{L}, \quad \bar{T} = \frac{T}{T_{scale}} \quad (103)$$

it turns into the following

$$\frac{\partial \bar{T}}{\partial \bar{t}} = \left(\frac{\delta D T T_{scale}^n}{mL^2} \right) \frac{\partial^2 \bar{T}^{n+1}}{\partial \bar{x}^2} \quad (104)$$

Setting the term in parentheses to unity automatically determines T .

$$T = \frac{(n+1)L^2}{\delta DT_{scale}^n} \quad (105)$$

So

$$\bar{\tau} = \frac{\tau}{T} = \frac{\tau \delta DT_{scale}^n}{(n+1)L^2} \quad (106)$$

Similarly, scaling Equation (101) leads to

$$\frac{\partial}{\partial \bar{t}} (\bar{T}_m \bar{T}_1) = \left[\frac{T \delta D \bar{T}_1^n T_{scale}^n}{L^2} \right] \frac{\partial^2}{\partial \bar{x}^2} (\bar{T}_m \bar{T}_1) \quad (107)$$

where \bar{T}_1^n is the scaled initial state of the temperature. Note that the term in brackets will be known as D_{eff} . Essentially what is left is

$$\frac{\partial}{\partial \bar{t}} f = D_{eff} \frac{\partial^2}{\partial \bar{x}^2} f \quad (108)$$

This, of course, is the linear diffusion equation (Equation 2) whose solution has already been found (Equation 18) which can be written as

$$f = \sum A_j e^{-\bar{t}/\bar{\tau}_j} \sin\left(\frac{j\pi \bar{x}}{x_{max}}\right) \quad (109)$$

where $\bar{\tau}_j$ is similar to Equation (19) with

$$\bar{\tau}_j = \frac{x_{max}^2}{j^2 \pi^2 D_{eff}} \quad (110)$$

and equivalent to the reciprocal of λ_n in Equation (89). Looking at the form of one of the modes

$$f = e^{-\bar{t}/\bar{\tau}_j} \sin\left(\frac{j\pi \bar{x}}{x_{max}}\right) \quad (111)$$

and taking the temporal and spatial derivatives

$$\frac{\partial f}{\partial t} = -\frac{1}{\tau_j} f \quad (112)$$

$$\frac{\partial^2 f}{\partial x^2} = -\frac{j^2 \pi^2}{x_{\max}^2} f \quad (113)$$

the timescale $\bar{\tau}_j$ is found to be (setting $j = 1$ since the lowest order mode has the most influence when the solution approaches the steady-state)

$$\bar{\tau}_1 = \frac{x_{\max}^2}{\pi^2 D_{\text{eff}}} \quad (114)$$

After expanding all terms and inserting D_{eff} and setting $\bar{x}_{\max} = 1$ the timescale of interest is

$$\bar{\tau}_1 = \frac{1}{\pi^2 (n+1) \bar{T}_0^n} \quad (115)$$

which is equivalent to the terms mentioned earlier from Equations (19) and (89). Again, this timescale is of interest because the lowest order mode has the most influence on the solution as it approaches steady-state. This allows for calculation of time-constants that let us know how fast a particular profile (be it a population or temperature profile) will approach the known steady-state solution. Note that \bar{T}_0^n is the average (the sum of the temperature over all the spatial points divided by the total number of spatial points then raised to the n^{th} power, $\langle \bar{T}_0 \rangle^n$) temperature of the steady-state distribution (this is explained in the next section and seen in [Figure 14](#)). Also, this is the theoretical prediction of the rate at which the temperature profile (already close to steady-state) will approach the steady-state.

4.6.2 Results of the Numerical Analysis of the Nonlinear Diffusion Equation

When looking to see how the timescale of the initial condition (see [Figure 2](#)) of interest behaves with nonlinearity a comparison was made between a modified version of Equation (115) (Equation 116) and the timescale values calculated from the numerical results $(\pi^2 \bar{\tau}_1)$. These values were calculated by looking at the different modes that are present in the solution. Since the steady-state solution is separate from the time-dependent portion of the solution in Equation (92) all that is of interest is the decay rates of the different amplitudes present,

$$\frac{1}{(n+1) \langle \bar{T}_c \rangle^n}, \quad (116)$$

The above quantity is plotted for values of n ranging from 0 to 10 and is compared to the values calculated from the numerical results. The timescale values are calculated when the temperature profile is close to steady-state and exhibit linear decay when plotted on a natural log scale.

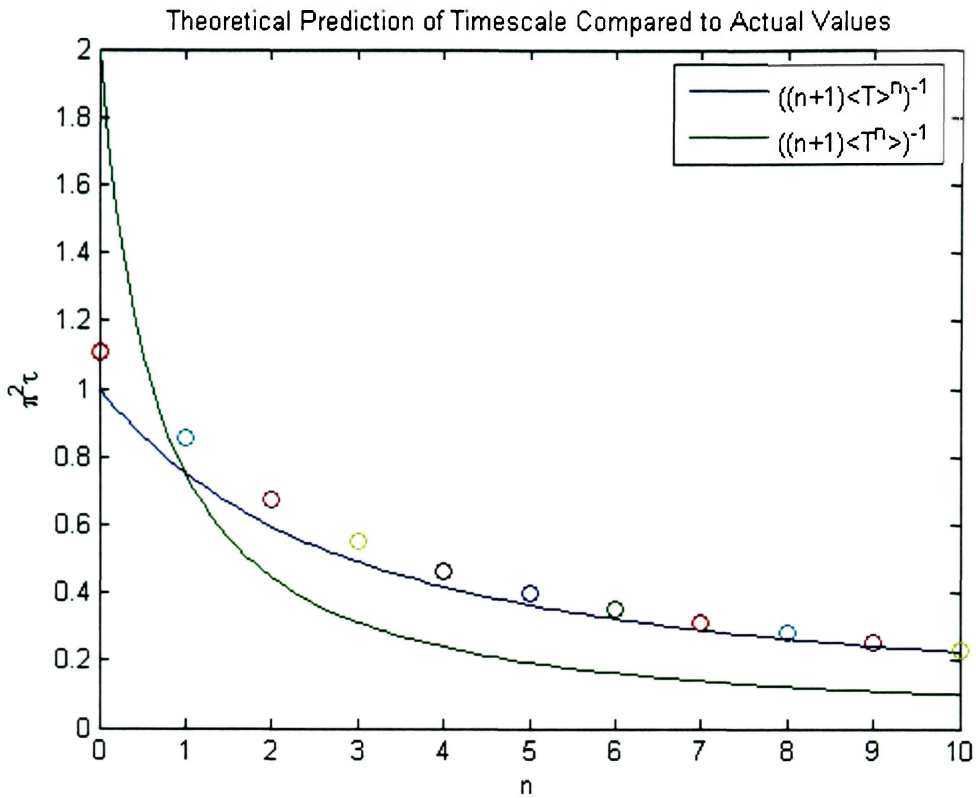


Figure 14. Theoretical Prediction of Timescale Compared to Actual Values.

Figure 14 shows how closely the calculated timescales follow the theoretical prediction. It is important to note that these timescales are calculated when the profile is very close to the steady-state solution (see Figure 15). The value for the timescale is the value of decay for every mode present in the solution once the wavefront nears the steady-state solution (see Figure 17).

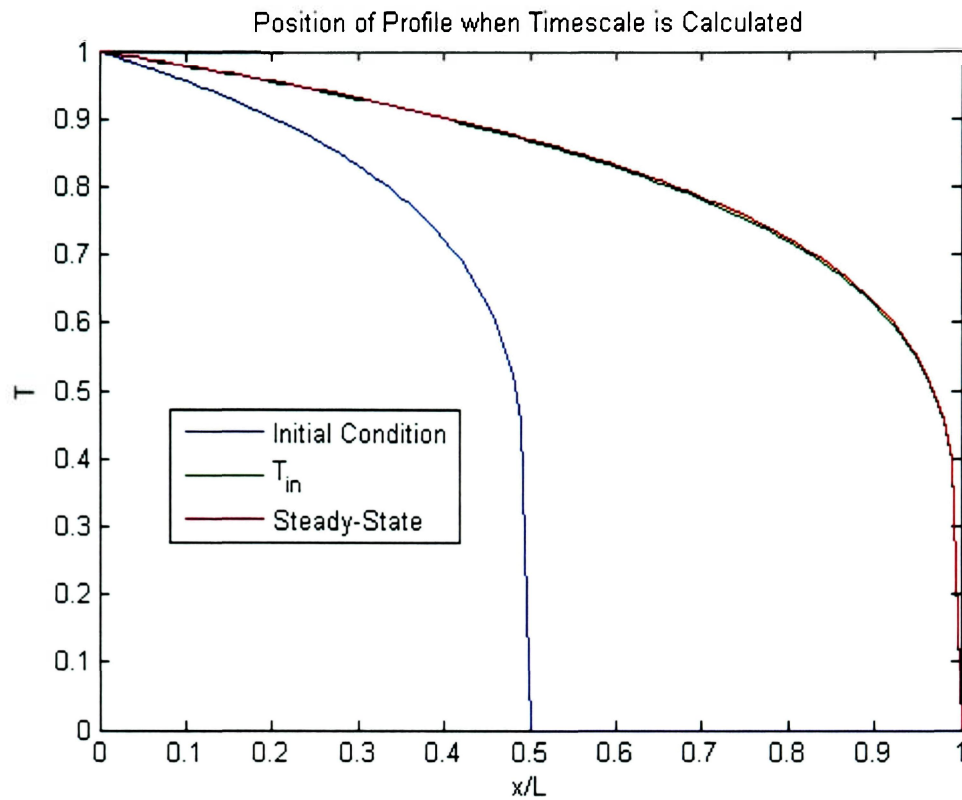


Figure 15. Position of Profile when Timescale is Calculated.

As can be seen in [Figure 15](#) the numerical solution is very close to the steady-state solution when the timescales are calculated. In this case $T_1 \ll T_m$. Also, T_m happens to be close to the steady-state solution (T_{ss}), but that simply helps in the evaluation of the timescales. The above figure represents the case where $n = 4$, which was chosen arbitrarily to show the nonlinear behavior of the system. In order to get the timescales out of the solution, the different modes had to be separated.

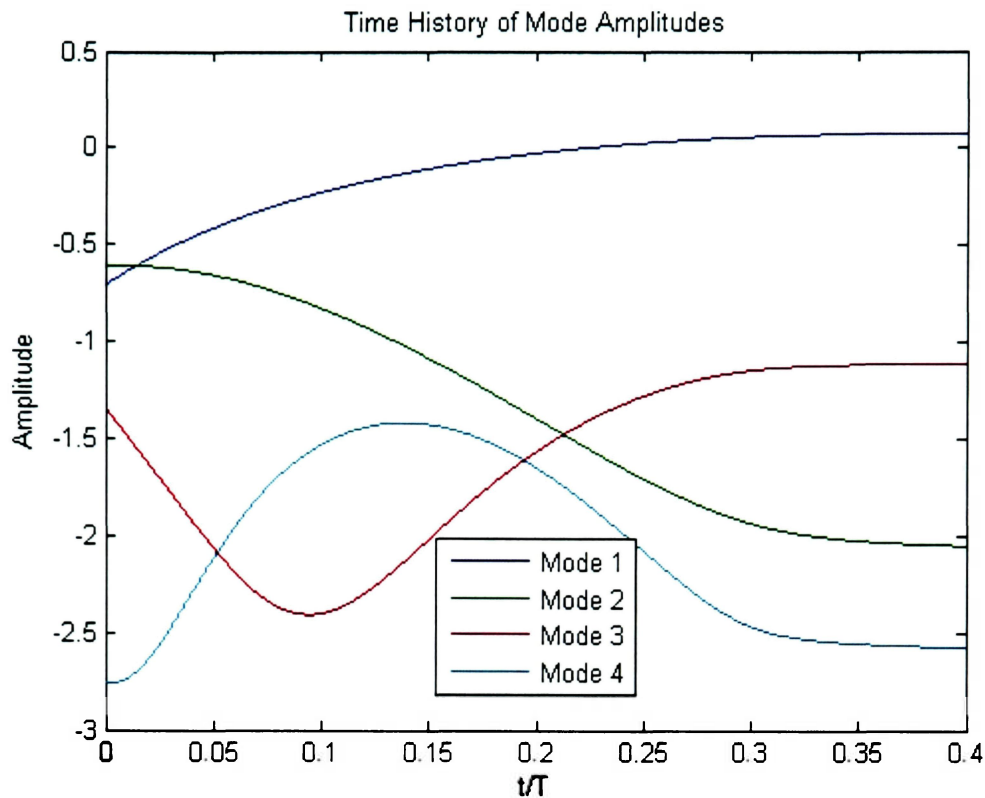


Figure 16. Time History of Mode Amplitudes.

Figure 16 shows the behavior of the different modes present in the temperature profile. It is important to note that the even modes eventually decay away while the odd modes increase to some steady-state value. This is related to Equation (92) since the steady-state is subtracted out of the solution only the odd modes of the solution should be present (as explained in **Section 4.5**). This is vastly different than the decay seen in Figure 6. This difference shows how this nonlinear diffusion equation is fundamentally different. We want to know how fast these modes approach their respective steady-state values. For these purposes the modes are calculated after subtracting out the steady-state solution from the numerical result.

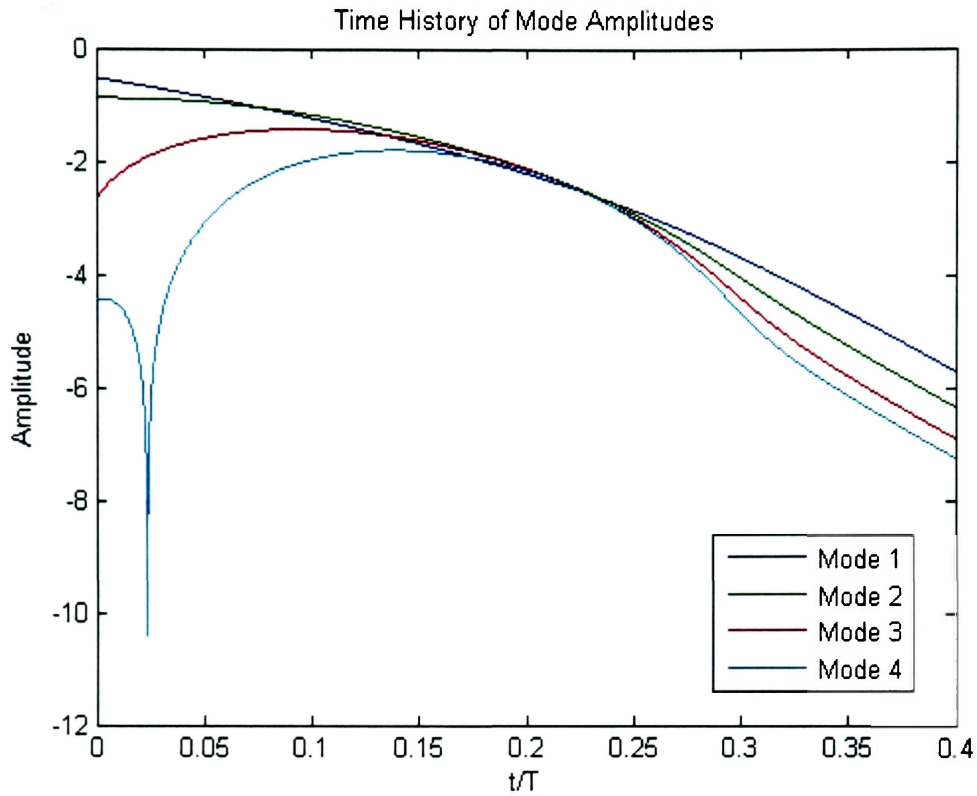


Figure 17. Time History of Mode Amplitudes.

Figure 17 shows the time history of the mode amplitudes ($n = 4$). Also, the above plot is that of $b_n e^{-\lambda_n^2 t}$ found in Equation (92) where the known steady-state solution is subtracted out from the numerical solution ($T - T_{ss}$) when the mode amplitudes are calculated. It is important to note that when the profile nears the steady-state solution ($\bar{t} > 0.35$), all of the modes decay at the same rate and exhibit linear behavior as desired. This is a very interesting phenomenon that needs further exploration.

4.6.3 Relation to Interstellar Population Expansion

As the order of nonlinearity increases, the leading edge of the profile becomes steeper (see Equation 80). This suggests self-similarity in the solutions [Newman and Sagan, 1981], which means that the shape of the temperature profile is preserved in the solution. This is true when the solution is not close to the right-hand boundary.

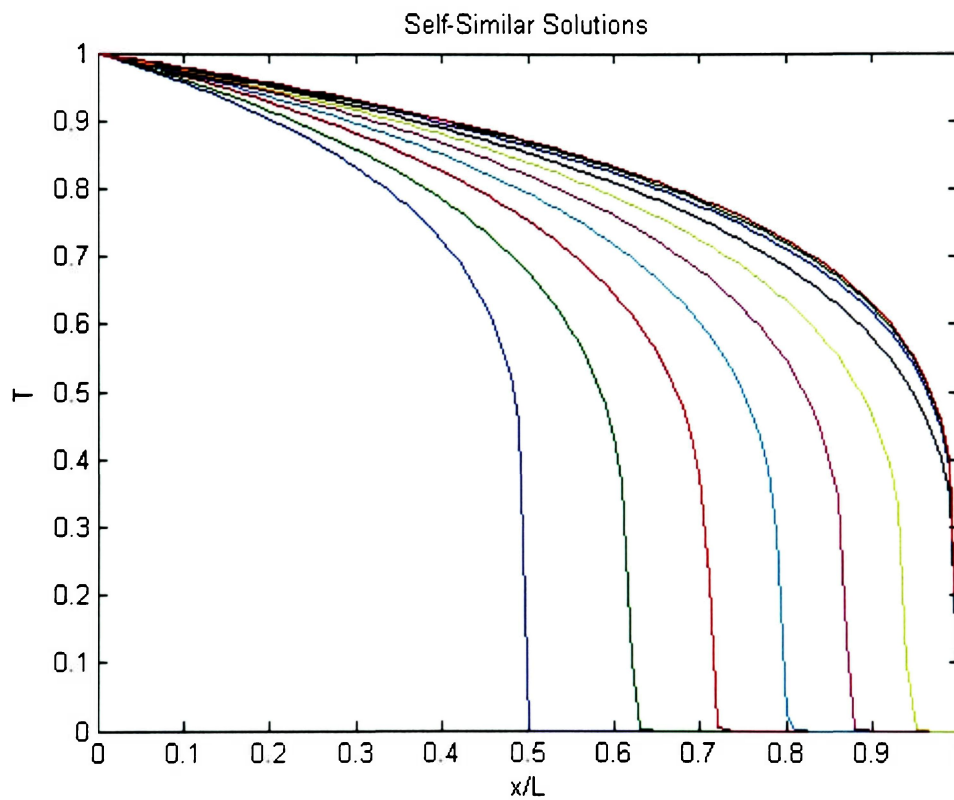


Figure 18. Self-Similar Solutions of the Expanding Wavefront ($n = 4$).

Figure 18 shows the self-similarity of the solution. What is meant by this is that each line shown in the figure represents the corresponding steady-state solution for that particular value of $T(\bar{x}) = 0$.

This leading edge can be thought of as the Population Expansion Front. Since this leading edge is so well-defined its velocity is relatively easy to calculate numerically. First, a specific value of Temperature (or population) is chosen. For the purposes of this thesis the value of 0.1 is chosen as the point to follow. Due to the nature of the solution, and the fact that the graph consists of a number of finite points, an algorithm was written to pick out the values of 0.1 and where they occur (See Appendix B: Code WavefrontVelocity.m).

This algorithm picks out those points and then plots their position as a function of time.

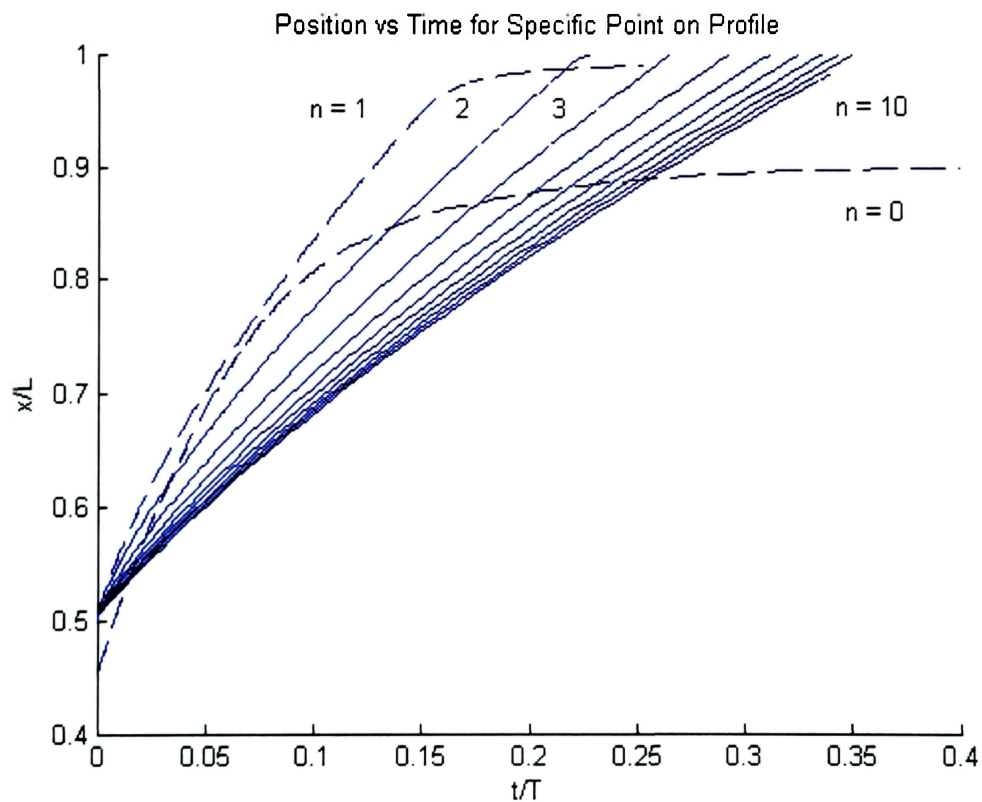


Figure 19. Position vs. Time Graph in Order to see Velocity of Expansion Wavefront.

Figure 19 shows the position as a function of time for the 0.1 Temperature point. The two dashed lines in the figure correspond to the $n = 0$ and $n = 1$ values for nonlinearity. These two values are dashed because they do not behave the same as the higher orders of nonlinearity (see Newman and Sagan, [1981]). The order of each solid line follows from $n = 2$ having the highest average slope and $n = 10$ having the lowest average slope.

What can be concluded from the above graph is that as the value of nonlinearity increases it takes the profile a longer time to reach the equilibrium state. This means that the profile travels slower as it becomes more nonlinear. The physical reason for this is that with the nonlinearity, it becomes harder for the “population” to expand. The power-law dependence of the diffusion coefficient slows down the diffusion process. More obstacles are getting in the way. This corresponds to results arrived at by Newman and Sagan (1981).

This is related to the decay constants calculated in Figure 14 and in Equation (116). Many factors could be attributed to this result. It tells us that when the population of a given species expands according to nonlinear diffusion, the higher the nonlinearity, the different array of factors that contribute to an individual's need to expand. These factors could be one of many: technological advances, the capacity of a planet to support life, how much space does an individual need to survive, or the sheer distance between habitable planets. Technological advances such as better space travel or increased life expectancy would decrease n making it easier for the population to expand. The farther

the distance a population would have to travel would increase the value of n making it more difficult to reach the next habitable planet.

5. Conclusions

Nature itself is nonlinear and having a better understanding of this nonlinearity is important. As can be seen from the above thesis the nonlinear diffusion equation is a marvelous equation that deserves greater attention.

From my research I have learned that nonlinear science is very complex. The nonlinear diffusion equation (solved by Newman and Sagan [1981]) is tricky to solve analytically; it has to be solved numerically. This brings up new challenges in deciding the best method to solve the equation. While there is no single correct way to solve this type of an equation the method of choice, if done properly, will expand the knowledge of the dynamics of this equation.

There is always error associated with any numerical analysis. The problem with this is just how much error is realistic to keep. There are ways to minimize the error or eliminate it completely, but for ease of computation and timeliness, a little error is always present, and helps divulge secrets about the nonlinear diffusion equation that are of great importance.

The main goal of this thesis was to determine the timescales over which the diffusion equation expands out into space, and more precisely, how fast does an initial temperature

profile approach a known steady-state solution. This allows us to compare to Newman and Sagan (1981) and their results on intergalactic population expansion. Theoretical predictions were made via exact analytical solutions and then compared to the numerical results. Several assumptions were made in order to simplify these calculations. For one, the equation was linearized leaving only one main component in the temperature profile allowing for analytical solutions to be derived with ease. This meant that we had to assume that the difference in temperature from one time-step to the next was insignificant. This might not necessarily be the case in Nature.

Different phenomena occur as the diffusion equation expands from the linear case to the nonlinear. For one, the curvature of the steady-state solution increases when the boundary conditions are non-zero. This is shown in [Figure 12](#). Second, when a source term is present, it takes much longer for the solution to reach the steady-state as the nonlinearity increases. This is related to [Figure 19](#) in that the initial condition can be thought of as having a source term for $\bar{x} < 1/2$ and as the nonlinearity increases the average velocity of the wavefront decreases. Thirdly, the particular case of interest leads to self-similar solutions that develop timescales independent of spatial or temporal scales. This means that the shape of the temperature profile is preserved in the solution. All modes that are present in the solution decay at the same rate as the temperature profile approaches the steady-state solution. This is due to the fact that the steady-state portion of the solution is subtracted out of the numerical solution.

Finally, in the case of interstellar population expansion, as the order of nonlinearity increases the rate at which the population expansion wavefront expands decreases. This is related to the decay constants calculated in [Figure 14](#) . Many factors could be attributed to this result. It tells us that when the population of a given species expands according to nonlinear diffusion, the higher the nonlinearity the slower the process. Several different factors contribute to an individuals' need to expand. Technological advances such as better space travel or increased life expectancy would decrease n making it easier for the population to expand. The farther the distance a population would have to travel would increase the value of n making it more difficult to reach the next habitable planet.

The main conclusion is that theoretical predictions were made and then verified by numerical means. This shows that numerical simulation is a valid technique that can be used to analyze complex physical phenomenon such as diffusion. Knowing that assumptions can be made in order to simplify calculations is a powerful tool. From these simple assumptions conclusion can be drawn about the nature of diffusion, and future research can expound upon these ideas to divulge further into the complex phenomenon known as diffusion.

References

- Chapra, Steven C., and Raymond P Canale. Numerical Methods for Engineers. New York: McGraw-Hill, 2002.
- Chen, Francis F. Introduction to Plasma Physics and Controlled Fusion. New York: Plenum Press, 1984.
- Fell, Andy. "New Models of Weather Pattern." UC Davis News & Information. Dec. 2005. University of California, Davis. 16 April 2006
http://www.news.ucdavis.edu/search/news_detail.lasso?id=7561.
- Hurricane Alley. "Hurricane Models Information." Hurricane Alley.net. 2005. HurricaneAlley.net. 16 April 2006 <http://www.hurricanealley.net/hurmdls.htm>.
- Mandelis, A. "Diffusion waves and their uses." *Phys. Today*. August 2000. page 29.
- Newman, William I., and Carl Sagan. "Galactic Civilizations: Population Dynamics and Interstellar Diffusion." Icarus Volume 46 (1981): Pages 293-327.
- Press, William H., Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. Numerical Recipes. New York: Cambridge University Press, 1987.

Smolyakov, A. I., and I. Khabibrakhmanov. "Nonlinear Diffusion of the Magnetic Field in Weakly Ionized Plasmas." Physical Review Letters Volume 81, Number 22 (1998).

Appendix A: Additional Derivations

A.1 Calculating the Stability of the Explicit Method (von Neumann stability analysis)

In order to determine the stability condition let T_j^i equal f_j^i and T_{j+1}^i and T_{j-1}^i equal $f_j^i e^{ik\Delta x}$ and $f_j^i e^{-ik\Delta x}$ respectively. This is known as von Neumann stability analysis [Chapra and Canale, 2002]. Making the previously mentioned substitutions for T into the finite difference equation leads to the following.

$$f_j^{i+1} = f_j^i + \frac{\Delta t}{(\Delta x)^2} (f_j^i e^{ik\Delta x} - 2f_j^i + f_j^i e^{-ik\Delta x}) \quad (117)$$

Manipulating this equation and making use of the trigonometric identities $\cos x = (e^{ix} + e^{-ix})/2$ and $\sin^2 a = 1 - \cos(2a)/2$ the equation can be rewritten as the following.

$$f_j^{i+1} = f_j^i \left[1 - \frac{4\Delta t}{(\Delta x)^2} \sin^2 \left(\frac{k\Delta x}{2} \right) \right] \quad (118)$$

From this form of the equation it can be seen that for the solution to converge the quantity in brackets, called the amplification factor, has to be less than 1 and greater than -1. The former is trivial to satisfy since all quantities in brackets are greater than one. The latter condition when satisfied leads to the stability condition, Equation (38), for the explicit method.

A.2 Adding a Source Term

Instead of setting the boundary temperature of the profile one could use a source term in the equation to provide the energy, or population growth needed to sustain the shapes of the temperature profiles. Adding in the source term changes Equation (60) into the following

$$\frac{\partial T}{\partial t} = \frac{\delta}{(n+1)} \frac{\partial^2 T^{n+1}}{\partial x^2} + Q(x,t), \quad (119)$$

where Q is the energy per unit volume per unit time added to the “environment”. Here we assume that the variable T actually represents the thermal energy density (energy-per-unit volume). We assume that Q is spatially localized—in practice a Dirac δ -function works well. This represents the physical situation in which the source term is a constant point source. The finite difference equation (Equation (68)) then becomes

$$\begin{aligned} & -\lambda(n+1)\left(\bar{T}'_{j+1}\right)^n \Delta \bar{T}_{j+1} + \left(1 + 2\lambda(n+1)\left(\bar{T}'_j\right)^n\right) \Delta \bar{T}_j - \lambda(n+1)\left(\bar{T}'_{j-1}\right)^n \Delta \bar{T}_{j-1} \\ & = \lambda\left(\bar{T}'_{j+1}\right)^{n+1} - 2\lambda\left(\bar{T}'_j\right)^{n+1} + \lambda\left(\bar{T}'_{j-1}\right)^{n+1} + \Delta \bar{t}(Q(\bar{x}, \bar{t})) \end{aligned} \quad (120)$$

The LU decomposition algorithm can be utilized to solve Equation (120) as before, with Q added to the constant b (see Equation (73)). With a constant source term the algorithm can be run for a long time to obtain the steady-state solution. This can then be compared to the steady-state solution found analytically, Equation (80). This provides support for replacing the steady temperature boundary conditions with a source.

$$\frac{\delta}{(n+1)} \frac{\partial^2 T^{n+1}}{\partial x^2} + Q(x,t) = 0 \quad (121)$$

When looking at Equation (121), if

$$\frac{\partial T}{\partial t} = 0 \quad (122)$$

and for all x where

$$Q(x, t) = 0 \quad (123)$$

then as shown earlier

$$\frac{\partial^2}{\partial x^2} (T^{n+1}) = 0 \quad (124)$$

$$T^{n+1} = Ax + B \quad (125)$$

Assume that the units of T are energy per volume and the units of Q are energy per time per volume. Now let

$$Q_0 = \xi_0 \delta(x) \quad (126)$$

After multiplying by dx on both sides and integrating

$$\int Q_0 dx = \xi_0 \quad (127)$$

and in the discrete case (as in the numerical algorithm)

$$\xi_0 = Q_{\max} \Delta x \quad (128)$$

Taking Equation (121) and integrating, the constants A and B can be solved for.

$$A = -\frac{\xi_0 (n+1)}{2\delta} \quad (129)$$

and

$$B = T_L^{n+1} - AL \quad (130)$$

Plugging these results back into Equation (125) and rearranging terms

$$T_{ss}(x) = \left[T_L^{n+1} + (L-x) \frac{\xi_0 (n+1)}{2\delta} \right]^{1/n+1} \quad (131)$$

Assuming that the temperature at the source occurs at $x=0$, Equation (131) can be rearranged to match Equation (80), which is the steady-state solution to the diffusion equation.

Now that the steady-state solution is known, a comparison can be made between the steady-state equation (Equation (131)) and what the LU decomposition algorithm will arrive at. For a value of $n=1$, the steady-state profile is shown in [Figure 20](#).

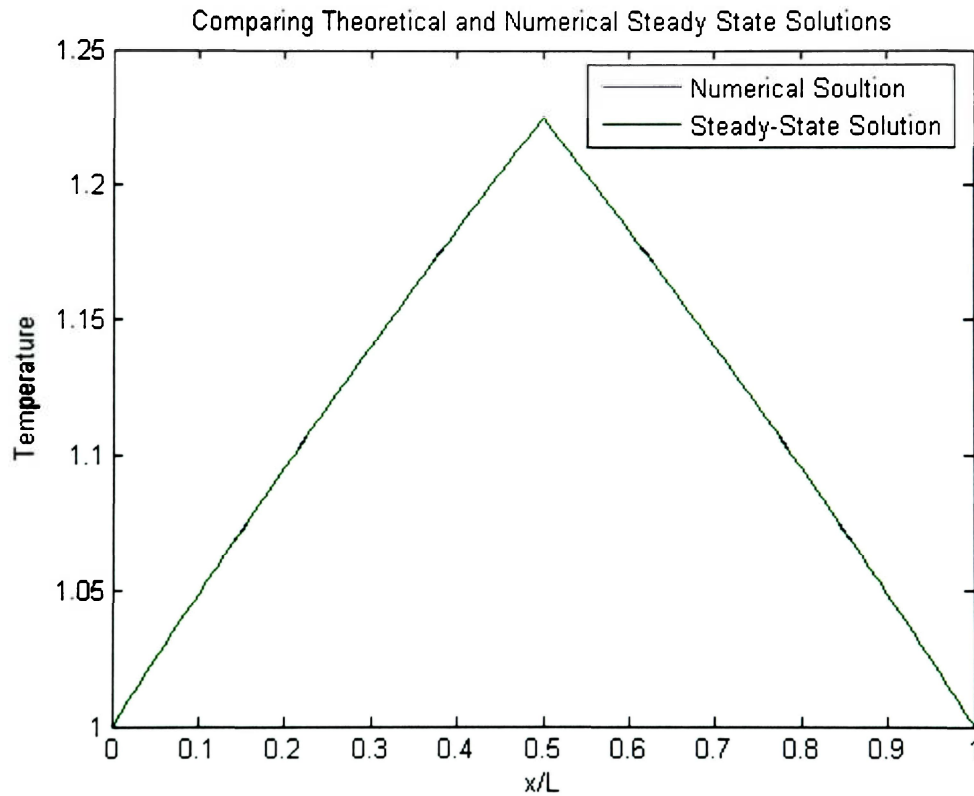


Figure 20. Steady-State Waveform for $n = 1$.

and the relative error between the two solutions is shown in [Figure 21](#).

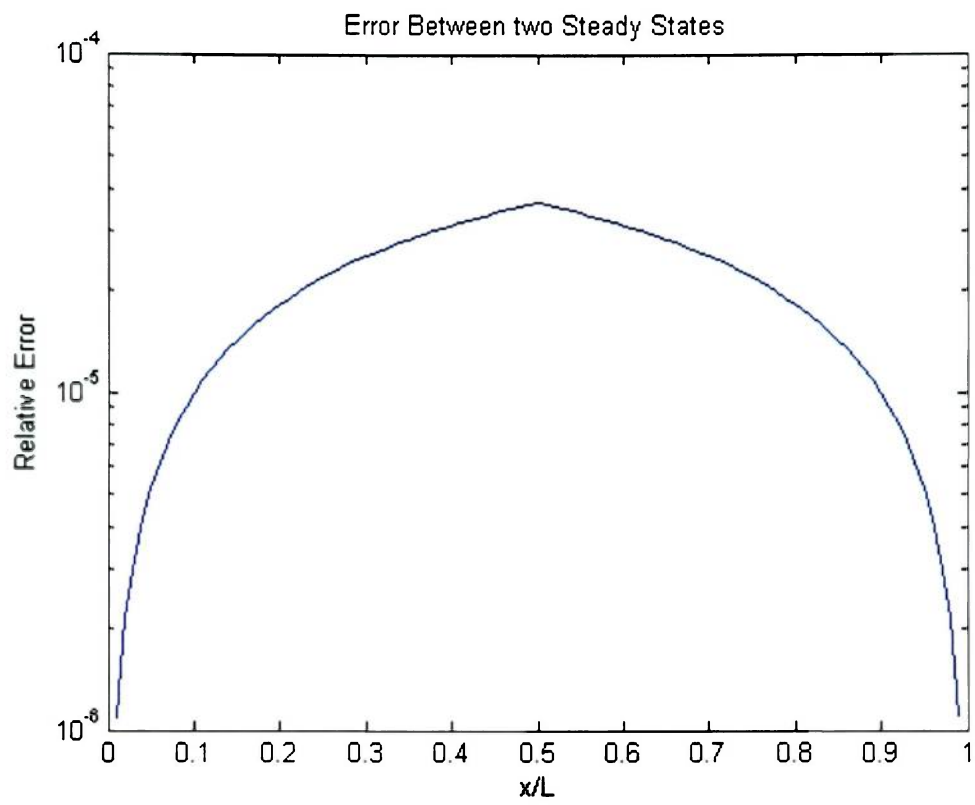


Figure 21. Relative Error between Numerical and Exact Steady-State Solutions.

Appendix B: Code

B.1 ConstantSourceGoodTimeNonlinearDiffusion.m

```
%Created by Dustin Sipka
%EP 700
%Time Dependent Nonlinear Diffusion Equation
%October 26, 2005
%Solves the time dependent nonlinear equation using the
implicit method and
%lets it run to get the steady state distribution and
compare it to the
%theoretical prediction

clear
clc
format long g

%Defining Initial Conditions and Constants
Diffusion = 1; %Diffusion coefficient
Length = 1; %Length of the space
TempL = 5; %Temp at end of space
Temp0 = 5; %Temp at beginning of space
Steady0 = 1.2247;
SteadyL = 1;
sLength = 0.5;
N = 1; %Constant factor for dimensionless
variables (decrease N decrease penetration depth, increase
N increase penetration depth)
n = 1; %Different values for nonlinear solution
space = 100; %Spatial resolution
sx = 0:sLength/(space):sLength;
sxbar = sx/sLength;
x = 0:Length/space:Length; %Number of spatial points
xbar = x/Length; %Unitless Space
dx = Length/space; %Spatial resolution
dxbar = dx/Length; %Unitless Space
w = 0.01*pi; %Frequency of boundary
temperature
time = 200000; %Number of time steps
t = 1:1:time; %Time step vector
tconstant = 1000; %Number of time constants that
elapse
T = (Length.^2*N)/(Diffusion*Temp0.^n); %Time constant
```

```

tbar = (1000)/(tconstant*T);%Unitless time
dt = dx.^2/2;                %Time resolution
dtbar = dt/tbar;             %Unitless time resolution
Temp = zeros(2,length(x)); %Size of temperature matrix
lambda = dtbar/((n+1)*dxbar.^2); %Coefficient in
numerical algorithm
Q = zeros(1,space+1);        %Source Term
Q(1,space/2) = 100;
periodtimesteps = round(2*pi/(w)); %Number of time steps
in one period of oscillation

%Initial Conditions-----
-----
%Solving for the initial Temperature distribution
for j = 1:length(sx)
    Steady1(j) = ((1/sLength)*(SteadyL.^(n+1)-
SteadyO.^(n+1))*sxbar(j)...
    +SteadyO.^(n+1)).^(1/(n+1));
end
for j = (space/2)+1:-1:1
    Steady2(52-j) = Steady1(j);
end
Steady3 = zeros(1,length(x));
for i = 1:51
    Steady3(i) = Steady2(i);
    Steady3(i+50) = Steady1(i);
end

Temp(1,:) = 5;
Temp(2,1) = 5;
Temp(2,length(x)) = 5;

%Scaling Temperature to Temp0
Temp = Temp/Temp0;

%Nonlinear LU Decomposition-----
-----
%Note: boundary nodes are ignored for this computation.
Decompositions
%starts at j = 2 and ends at j = length(x)-1

%Filling Three Vectors of the Tridiagonal Matrix
d = zeros(size(x));
u = zeros(size(x));
l = zeros(size(x));
%Note: u(space-1)=0 and l(1) = 0

```

```

D = zeros(size(d));           %Intermediate vector for LU
decomposition
deltaTemp = zeros(size(Temp));
deltaTemp(:,1) = 0;
deltaTemp(:,length(x)) = 0;

h = waitbar(0,'There is no spoon...');
for m = 2:time
    waitbar(m/time)
    for i = 1:space-1

        %Setting up B
        B(1,i) = lambda*(Temp(1,i)).^(n+1)-
2*lambda*(Temp(1,i+1)).^(n+1)+lambda*(Temp(1,i+2)).^(n+1) +
dtbar*Q(1,i);

        %Setting up the three diagonal vectors
        d(i) = (1+2*(n+1)*lambda*(Temp(1,i)).^n);
        if (i == 1)
            l(i) = 0;
        else
            l(i) = -1*lambda*(n+1)*(Temp(1,i-1)).^n;
        end
        if(i == space-1)
            u(i) = 0;
        else
            u(i) = -1*lambda*(n+1)*(Temp(1,i+1)).^n;
        end

        %Decomposition of coefficient matrix
        factor = l(i+1)/d(i);
        l(i+1) = factor;
        d(i+1) = d(i+1) - factor*u(i);

        %Forward Substitution
        if(i == 1)
            D(i) = B(1,i);
        else
            D(i) = B(1,i) - l(i)*D(i-1);
        end
    end
end
for i = space:-1:2
    %Backward substitution
    if(i == space)
        deltaTemp(1,i) = D(i-1)/d(i-1);
    else

```

```

                deltaTemp(1,i) = (D(i-1) - u(i-
1)*deltaTemp(1,i+1))/d(i-1);
            end
        end

        %Solving for the new temperatures
        for i = 2:space
            Temp(2,i) = Temp(1,i) + deltaTemp(1,i);
            Temp(1,i) = Temp(2,i);
        end
    end
end
close(h);

Error = abs(Temp(2,:)-Steady3)./Steady3;

figure(1)
plot(x,Temp(2,:),x,Steady3);
xlabel('xbar');
ylabel('Temperature');
title('Comparing Theoretical and Numerical Steady State
Solutions');
legend('Numerical Souttion','Steady-State Solution');

figure(2)
semilogy(x,Error);
xlabel('xbar');
ylabel('Relative Error');
title('Error Between two Steady States');

```

B.2 FinalAlgorithmNonlinear.m

```

%Created by Dustin Sipka
%EP 700
%Time Dependent Nonlinear Diffusion Equation
%January 30, 2006
%Solves the time dependent nonlinear equation using the
implicit method and
%lets it run to get the steady state distribution and
compare it to the
%theoretical prediction, also puts into one file all of the
analysis that
%has been done so results can be arrived at easily

clear
clc

```

```

format long g

%Defining Initial Conditions and Constants
Diffusion = 1; %Diffusion coefficient
Length = 1; %Length of the space
TempL = 1; %Temp at end of space
Temp0 = 1; %Temp at beginning of space
sLength = 0.5; %Length of steady-state
solution for constant source
sLength2 = 1; %Length of steady-state
solution with constant null
N = 1; %Constant factor for
dimensionless variables (decrease N decrease penetration
depth, increase N increase penetration depth)
n = 5; %Different values for
nonlinear solution
space = 100; %Spatial resolution
sx = 0:sLength/(space):sLength; %Spatial resolution for
steady-state
sxbar = sx/sLength; %Unitless space for steady-
state
x = 0:Length/space:Length; %Number of spatial points
xbar = x/Length; %Unitless space
dx = Length/space; %Spatial resolution
dxbar = dx/Length; %Unitless space
w = 0.01*pi; %Frequency of source
time = 4000; %Number of time steps
t = 1:1:time; %Time step vector
tconstant = 1000; %Number of time constants
that elapse
T = (Length.^2*N)/(Diffusion*Temp0.^n); %Time constant
tbar = (1000)/(tconstant*T); %Unitless time
dt = dx.^2/2; %Time resolution
dtbar = dt/tbar; %Unitless time
resolution
Temp = zeros(time,length(x)); %Size of
temperature matrix
lambda = dtbar/((n+1)*dxbar.^2); %Coefficient in
numerical algorithm
Q = zeros(time,space+1); %Source Term
Q(1,space/2) = 100; %source Term
periodtimesteps = round(2*pi/(w)); %Number of time
steps in one period of oscillation
SteadyL = 1; %Temp at end for
steady-state solution

```

```

Steady0 = (SteadyL.^(n+1) + (Length-
0.5).*((Q(1,space/2)*dxbar*(n+1))/(2*Diffusion))).^(1/(n+1)
);    %Temp at source for steady-state solution

%Initial Conditions-----
-----

%Solving for the initial Temperature distribution
%Calculates the initial temperature distribution
for j = 1:length(sx)
    Steady1(j) = ((1/sLength)*(SteadyL.^(n+1)-
Steady0.^(n+1))*sxbar(j)...
    +Steady0.^(n+1)).^(1/(n+1));
end

%Calculates the steady-state for distribution with constant
null
for j = 1:length(sx)
    SteadySS(j) = ((1/sLength2)*(SteadyL.^(n+1)-
Steady0.^(n+1))*sxbar(j)...
    +Steady0.^(n+1)).^(1/(n+1));
end

%For distribution with extended null area
for i = 1:51
    Steady3(i) = Steady1(i);
    Steady3(i+50) = SteadyL;
end

Temp(1,:) = Steady3;

%Putting in the boundary conditions for all time
for i = 1:time
    Temp(i,1) = Temp(1,1);
    Temp(i,length(x)) = Temp(1,length(x));
%    Q(i,space/2) = 100;          %Steady source
%    Q(i,(space/2)) = Q(i,space/2) + 100*sin(w*(i-1));
%Oscillating source with steady source
end

%Scaling Temperature to Temp0
%Temp = Temp/Temp(1,1);          %For symmetric
distribution with constant source
Temp = Temp/Temp(1,length(x));  %For distribution with
extended null area

%Nonlinear LU Decomposition-----
-----

```

```

%Note: boundary nodes are ignored for this computation.
Decompositions
%starts at j = 2 and ends at j = length(x)-1

%Filling Three Vectors of the Tridiagonal Matrix
d = zeros(size(x));
u = zeros(size(x));
l = zeros(size(x));
%Note: u(space-1)=0 and l(1) = 0

D = zeros(size(d));           %Intermediate vector for LU
decomposition
deltaTemp = zeros(size(Temp));
deltaTemp(:,1) = 0;
deltaTemp(:,length(x)) = 0;

h = waitbar(0,'There is no spoon...');
for m = 2:time
    waitbar(m/time)
    for i = 1:space-1

        %Setting up B
        B(m-1,i) = lambda*(Temp(m-1,i)).^(n+1) -
2*lambda*(Temp(m-1,i+1)).^(n+1)+lambda*(Temp(m-
1,i+2)).^(n+1);% + dtbar*Q(m-1,i);

        %Setting up the three diagonal vectors
        d(i) = (1+2*(n+1)*lambda*(Temp(m-1,i)).^n);
        if (i == 1)
            l(i) = 0;
        else
            l(i) = -1*lambda*(n+1)*(Temp(m-1,i-1)).^n;
        end
        if(i == space-1)
            u(i) = 0;
        else
            u(i) = -1*lambda*(n+1)*(Temp(m-1,i+1)).^n;
        end

        %Decomposition of coefficient matrix
        factor = l(i+1)/d(i);
        l(i+1) = factor;
        d(i+1) = d(i+1) - factor*u(i);

        %Forward Substitution
        if(i == 1)
            D(i) = B(m-1,i);

```

```

        else
            D(i) = B(m-1,i) - l(i)*D(i-1);
        end
    end
    for i = space:-1:2
        %Backward substitution
        if(i == space)
            deltaTemp(m-1,i) = D(i-1)/d(i-1);
        else
            deltaTemp(m-1,i) = (D(i-1) - u(i-
1)*deltaTemp(m-1,i+1))/d(i-1);
        end
    end
    end

    %Solving for the new temperatures
    for i = 2:space
        Temp(m,i) = Temp(m-1,i) + deltaTemp(m-1,i);
    end
end
close(h);

%Plots the Different Time Steps as a movie-----
-----
%mov = avifile('difmov.avi','fps',10);
for i = 1:time/10
    figure(1);
    plot(x,Temp(10*i,:),x,SteadySS);
    title('General Nonlinear Diffusion with Non-Zero
Boundary Conditions');
    xlabel('Length of Wave Solution');
    ylabel('Amplitude of Solution');
    %legend('Numerical Solution','Exact Solution');
    %axis([0 Length 0 2]);
    axis([0 Length 0.9 1.3]);
    F(i) = getframe;
    %mov = addframe(mov,F(i));
end
%mov = close(mov);
movie(F);    %Shows Plots as a Movie

```

B.3 FinalCrankNicolsonAlgorithmNonlinear.m

```

%Created by Dustin Sipka
%EP 700
%Time Dependent Nonlinear Diffusion Equation

```



```

%January 30, 2006
%Solves the time dependent nonlinear equation using the
implicit method and
%lets it run to get the steady state distribution and
compare it to the
%theoretical prediction, also puts into one file all of the
analysis that
%has been done so results can be arrived at easily

clear
clc
format long g

%Defining Initial Conditions and Constants
%Parameters that can be controlled and changed
parameter = 1/2;           %Parameter for Crank-
Nicolson variation
%Diffusion = 1;           %Diffusion coefficient
Length = 1;               %Length of the space
%TempL = 1;               %Temp at end of space
%Temp0 = 1;               %Temp at beginning of space
SteadyL = 0;              %Temp at end for steady-
state solution
Steady0 = 1;              %Temp at beginning of
steady-state solution
sLength = 0.5;            %Length of steady-state
solution for constant source
sLength2 = 1;             %Length of steady-state
solution with constant null
%N = 1;                   %Constant factor for
dimensionless variables (decrease N decrease penetration
depth, increase N increase penetration depth)
n = 4;                    %Different values for
nonlinear solution
n2 = 0;
space = 100;              %Spatial resolution
%w = 0.01*pi;            %Frequency of source
time = 40000;             %Number of time steps
dtbar = 1e-5;            %Resolution of time

%Parameters the algorithm needs to know
sx = 0:sLength/(space):sLength; %Spatial
resolution for steady-state
sxbar = sx/sLength;      %Unitless
space for steady-state
x = 0:Length/space:Length; %Number of
spatial points

```

```

xbar = x/Length; %Unitless
space
%dx = Length/space; %Spatial
resolution
dxbar = Length/space; %Unitless
space
%T = (Length.^2*N*(n+1))/(Diffusion*Temp0.^n); %Time
constant
%dtbar = dt/T; %Unitless
time resolution
tbar = time*dtbar; %Unitless
time
tbarv = 0:dtbar:tbar; %Unitless
time vector

%Parameters for analysis
Temp = zeros((time/10),length(x)); %Size of
temperature matrix
TempF = zeros(10,length(x)); %Size of
algorithm matrix
lambda = dtbar/(dxbar.^2); %Coefficient in
numerical algorithm
%Q = zeros(time,space+1); %Source Term
%Q(1,space/2) = 100; %Source Term
%periodtimesteps = round(2*pi/(w)); %Number of time
steps in one period of oscillation
t = 1:1:(time/10); %Time step
vector
t = t*10*dtbar; %tbar vector
%Steady0 = (SteadyL.^(n+1) + (Length-
0.5).*((Q(1,space/2)*dxbar*(n+1))/(2*Diffusion)).^(1/(n+1)
)); %Temp at source for steady-state solution

%Initial Conditions-----
-----
%Solving for the initial Temperature distribution
%Calculates the initial temperature distribution
for j = 1:length(sx)
    Steady1(j) = ((1/sLength)*(SteadyL.^(n+1)-
Steady0.^(n+1))*sxbar(j)...
    +Steady0.^(n+1)).^(1/(n+1));
% Steady1(j) = 0;
end

%Calculates the steady-state for distribution with constant
null
for j = 1:length(sx)

```

```

        SteadySS(j) = ((1/sLength2)*(SteadyL.^(n+1)-
SteadyO.^(n+1))*sxbar(j)...
        +SteadyO.^(n+1)).^(1/(n+1));
end

%For distribution with extended null area
for i = 1:((space/2)+1)
    Steady3(i) = Steady1(i);
    Steady3(i+(space/2)) = SteadyL;
end

Temp(1,:) = Steady3;
%Temp(1,1) = 1;

%Putting in the boundary conditions for all time
for i = 0:((time/10)-1)
    Temp(i+1,1) = Temp(1,1);
    Temp(i+1,length(x)) = Temp(1,length(x));
%    Q(i,space/2) = 100;           %Steady source
%    Q(i,(space/2)) = Q(i,space/2) + 100*sin(w*(i-1));
%Oscillating source with steady source
end

for i = 1:10
    TempF(i,1) = Temp(1,1);
    TempF(i,length(x)) = Temp(1,length(x));
end

%Nonlinear LU Decomposition-----
-----
%Note: boundary nodes are ignored for this computation.
Decompositions
%starts at j = 2 and ends at j = length(x)-1

%Filling Three Vectors of the Tridiagonal Matrix
d = zeros(size(x));
u = zeros(size(x));
l = zeros(size(x));
%Note: u(space-1)=0 and l(1) = 0

D = zeros(size(d));           %Intermediate vector for LU
decomposition
deltaTemp = zeros(size(TempF));
deltaTemp(:,1) = 0;
deltaTemp(:,length(x)) = 0;

h = waitbar(0,'There is no spoon...');

```

```

for q = 1:((time/10)-1)
waitbar(q/(time/10),h)
TempF(1,:) = Temp(q,:);
for m = 2:10
    for i = 1:space-1

        %Setting up B
        B(m-1,i) = lambda*parameter*(TempF(m-1,i)).^(n+1)-
2*lambda*parameter*(TempF(m-
1,i+1)).^(n+1)+lambda*parameter*(TempF(m-1,i+2)).^(n+1)+(1-
parameter)*lambda*(TempF(m-1,i).^(n+1)-2*TempF(m-
1,i+1).^(n+1)+TempF(m-1,i+2).^(n+1));% + dtbar*Q(m-1,i);

        %Setting up the three diagonal vectors
        d(i) = (1+2*(n+1)*lambda*parameter*(TempF(m-
1,i)).^n);
        if (i == 1)
            l(i) = 0;
        else
            l(i) = -1*lambda*parameter*(n+1)*(TempF(m-1,i-
1)).^n;
        end
        if(i == space-1)
            u(i) = 0;
        else
            u(i) = -1*lambda*parameter*(n+1)*(TempF(m-
1,i+1)).^n;
        end

        %Decomposition of coefficient matrix
        factor = l(i+1)/d(i);
        l(i+1) = factor;
        d(i+1) = d(i+1) - factor*u(i);

        %Forward Substitution
        if(i == 1)
            D(i) = B(m-1,i);
        else
            D(i) = B(m-1,i) - l(i)*D(i-1);
        end
    end
end
for i = space:-1:2
    %Backward substitution
    if(i == space)
        deltaTemp(m-1,i) = D(i-1)/d(i-1);
    else

```

```

        deltaTemp(m-1,i) = (D(i-1) - u(i-
1)*deltaTemp(m-1,i+1))/d(i-1);
    end
end

%Solving for the new temperatures
for i = 2:space
    TempF(m,i) = TempF(m-1,i) + deltaTemp(m-1,i);
end
end
Temp(q+1,:) = TempF(10,:);
end
close(h);

[a,b] = find(Temp > 0.099 & Temp < 0.100);

%Calculates error between steady-state and numerical
solution for
%initial condition with constant null area to see how fast
the
%numerical solution is approaching the steady-state
for m = 1:(time/10)
    for z = 1:space+1
        Speed(m,z) = SteadySS(z)-Temp(m,z);
    end
end

%This calculates the time constant
tau = abs((log(Speed((time/10)-5),51))-
log(Speed((time/10)-6),51)))/(1*10*dtbar);
time-constant = 1/tau;

%Fourier Transform to look at amplitudes of different modes
for initial
%condition with constant null region
clear q k j i

p = 0:1:10;      %Mode of Interest

%Subtracting out the steady-state solution
TempFour = zeros(size(Temp));

for k = 1:(time/10)
    for j = 1:space+1
        TempFour(k,j) = Temp(k,j) - SteadySS(j);
    end
end
end

```

```

%Fourier Transform
for q = 1:length(p)
    for k = 1:((time/10))
        for j = 1:space
            %Transform(j) =
TempFour(k, j)*exp(2*i*pi*p(q)*(j-1)/(space))/(space);
            Transform(j) =
TempFour(k, j)*sin(pi*p(q)*x(j))/(space);
        end
        Frequency(k) = 2*sum(Transform);
    end
    Amplitudes(q,:) = Frequency;
end
Amplitudes2 = sqrt(Amplitudes.*(Amplitudes));

%This calculates the time constant of the present sine
modes
tau2 = abs((log(Amplitudes2(2,((time/10)-2500)))-
log(Amplitudes2(2,((time/10)-2501)))))/(1*10*dtbar);
time-constant2 = 1/tau2;

%Plotting the Time History of the Mode Amplitudes
figure(6)
plot(t,log(Amplitudes2(2,:)),t,log(Amplitudes2(3,:)),t,log(
Amplitudes2(4,:)),t,log(Amplitudes2(5,:)));
xlabel('tbar');
ylabel('Amplitude');
legend('Mode 1','Mode 2','Mode 3','Mode
4','Location','Best');
title('Time History of Mode Amplitudes');

figure(7)
plot(x,Temp(1,:),x,Temp(time/10,:),x,SteadySS);
xlabel('xbar');
ylabel('Temp');
legend('Initial Condition','Numerical Solution','Steady-
State','Location','Best');
title('Position of Wavefront when Timescale is
Calculated');

figure(8)
plot(x,Temp(1,:),x,Temp(500,:),x,Temp(1000,:),x,Temp(1500,
),x,Temp(2000,:),x,Temp(2500,:),x,Temp(3000,:),x,Temp(3500,
:),x,Temp(4000,:),x,SteadySS);
xlabel('xbar');
ylabel('Temp');

```

```
title('Self-Similar Solutions');
```

B.4 GoodDiffusion.m

```
%Created by Dustin Sipka
%EP 700
%General Diffusion Equation
%April 6, 2005

clear
clc
format long g

%Note that in T(i,n) "i" represents time and "n" represents
space

%Definition of constants and parameters
order = 10;           %Order of Interest
diffusion = 1;       %Diffusion Coefficient
L = 1;               %Length of Wave
T = L.^2/diffusion;  %Makes Coefficient "1"
time = 100;          %Time Steps
tconstant = 10;      %Number of time constants that
elapse
tbar = (time/tconstant)/T; %Unitless Time
Steps
space = 500;         %Resolution of Solution
dx = L/space;        %Spatial resolution
dxbar = dx/L;        %Unitless Spatial Resolution
dt = ((L/(order*pi)).^2/diffusion); %Time
resolution
dtbar = dt/tbar;     %Unitless dt
theta = 0:L/space:L; %Values for Solution
xbar = theta/L;      %Unitless Space
lambda = (diffusion*dtbar)/(dxbar.^2); %Constant from
FDE
alpha = 1+2*lambda;  %Constant from FDE
Temp = zeros(time,space+1);
t = 0:dtbar:(99*dtbar);

%Solving Numerically Using LU Decomposition-----
-----
%Boundary Conditions
for i = 1:time
    Temp(i,1) = 0;
```

```

    Temp(i,space+1) = 0;
end

%Initial Conditions (b vector in LU Decomposition)
Temp(1,:) = sin(pi*xbar) + (1/1)*sin(2*pi*xbar)...
    + (1/1)*sin(3*pi*xbar) + (1/1)*sin(4*pi*xbar)...
    + (1/1)*sin(5*pi*xbar) + (1/1)*sin(6*pi*xbar)...
    + (1/1)*sin(7*pi*xbar) + (1/1)*sin(8*pi*xbar)...
    + (1/1)*sin(9*pi*xbar) + (1/1)*sin(10*pi*xbar);

%Filling Three Vectors of the Tridiagonal Matrix
for n = 1:space-1
    d(n) = alpha;
    u(n) = -lambda;
    l(n) = -lambda;
end
u(space-1) = 0;
l(1) = 0;
D = zeros(size(d)); %Intermediate Vector in LU
Decomposition

%Decomposition of Tridiagonal Matrix
for k = 2:space-1
    factor = l(k)/d(k-1);
    l(k) = factor;
    d(k) = d(k) - factor*u(k-1);
end

%Forward and Backward Substitution to Solve for Next Time
Step
for m = 2:time
    B = Temp;
    %Forward (only works for tridiagonal sparse matrix)
    for i = 1:space-1
        if (i == 1)
            B(m-1,i+1) = B(m-1,i+1) + lambda*B(m-1,i);
        elseif (i == space-1)
            B(m-1,i+1) = B(m-1,i+1) + lambda*B(m-1,i+2);
        end
        if (i == 1)
            D(i) = B(m-1,i+1); %First element of D
        else
            D(i) = B(m-1,i+1) - l(i)*D(i-1); %Forward
loop
        end
    end
end
%Backward (only works for tridiagonal sparse matrix)

```



```

    for i = space:-1:2
        if (i == space)
            Temp(m,i) = D(i-1)/d(i-1); %Last element of X
        else
            Temp(m,i) = (D(i-1) - u(i-1)*Temp(m,i+1))/d(i-
1); %Backward loop
        end
    end
end

%Performs Fourier Transform to Find Amplitudes of Different
Modes-----
for i = 1:time
    for m = 1:10
        for n = 1:space
            transform(n) =
Temp(i,n)*sin(m*pi*xbar(n))/(space);
        end
        Amplitude(i,m) = 2*sum(transform);
    end
end

%Solving the Exact Solution -----
-----
%Setting Up Time Constant
for n = 1:10
    tau(n) = (L/((n)*pi)).^2/diffusion;
end

dtexact = tau(order)/(time/tconstant);
texact = 0:dtexact:time*dtexact;

%Setting Up Amplitudes
for i = 1:10
    for n = 1:time
        A(n,i) = exp(-texact(n)/tau(i));
    end
end

%Exact Solution
%Boundary Conditions
for i = 1:time
    Ex(i,1) = 0;
    Ex(i,space+1) = 0;
end

%Solution

```

```

for i = 1:time
    for n = 2:space
        Ex(i,n) = A(i,1)*sin(pi*xbar(n)) +
A(i,2)*sin(2*pi*xbar(n))...
            + A(i,3)*sin(3*pi*xbar(n)) +
A(i,4)*sin(4*pi*xbar(n))...
            + A(i,5)*sin(5*pi*xbar(n)) +
A(i,6)*sin(6*pi*xbar(n))...
            + A(i,7)*sin(7*pi*xbar(n)) +
A(i,8)*sin(8*pi*xbar(n))...
            + A(i,9)*sin(9*pi*xbar(n)) +
A(i,10)*sin(10*pi*xbar(n));
    end
end

%-----
%-----
%Calculates Error Using RMS of Actual Solution to the
Numerical Solution of
%'order'
ErrorSol = (Temp - Ex).^2;
for i = 1:time
    RMS(i) = sqrt(sum(ErrorSol(i,:))/(space+1));
end

%Calculates Error Between Actual and Numerical Amplitudes
ErrorAmp = abs(A(:,order) - Amplitude(:,order));

%Plotting-----
%-----
x = 1:1:time; %Used to Plot Change in Error with Time

figure(1)
plot(theta,Temp(time,:),theta,Ex(time,:));
title('General Diffusion with Dirichlet Boundary
Conditions');
xlabel('Length of Wave Solution');
ylabel('Temperature');
legend('Numerical Solution','Exact Solution');

figure(2)
semilogy(t(1:100),RMS(1:100));
title('Error Between Solutions over Time');
xlabel('Number of Time Steps');
ylabel('Error Between Solutions');

figure(3)

```

```

semilogy(t,ErrorAmp);
title('Error Between Amplitudes of Solutions');
xlabel('tbar');
ylabel('Error Between Amplitudes');

figure(4)
plot(t,A(:,order),t,Amplitude(:,order));
title('Amplitude of Selected Order');
xlabel('tbar');
ylabel('Amplitude of Order');
legend('Exact Solution','Numerical Solution');

%Time Lapse Plot of Solution
figure(5)
subplot(2,2,1)
plot(theta,Temp(1,:));
xlabel('tbar = 1');
ylabel('Amplitude');
axis([0 L -1 8]);
subplot(2,2,2)
plot(theta,Temp(25,:));
xlabel('tbar = 25');
axis([0 L -1 8]);
subplot(2,2,3)
plot(theta,Temp(50,:));
xlabel('tbar = 50');
ylabel('Amplitude');
axis([0 L -1 8]);
subplot(2,2,4)
plot(theta,Temp(75,:));
xlabel('tbar = 75');
axis([0 L -1 8]);

figure(6)
plot(theta,Temp(1,:),theta,Temp(25,),'--',
theta,Temp(50,),':',theta,Temp(75,),'-.',
theta,Temp(100,),'-');
%plot(theta,Temp(1,:));
ylabel('Temp');
xlabel('Length of Solution');
title('Solution of General Diffusion Equation');
legend('tbar = 0','tbar = 25','tbar = 50','tbar = 75','tbar = 100');

figure(7)
plot(t,Amplitude);
ylabel('Amplitude');

```

```

xlabel('tbar');
title('Decay of Amplitudes Over Time');
axis([0 99*dtbar 0 1]);
legend('Mode 1','Mode 2','Mode 3','Mode 4','Mode 5','Mode 6',
'Mode 7','Mode 8','Mode 9','Mode 10');

figure(8)
plot(theta,sqrt(ErrorSol(100,:)));
xlabel('Length of Wave Solution');
ylabel('Absolute Error');
title('Absolute Error Relative to Position');

```

B.5 GoodTimeNonlinearDiffusion.m

```

%Created by Dustin Sipka
%EP 700
%Time Dependent Nonlinear Diffusion Equation
%September 23, 2005
%Solves the time dependent nonlinear equation using the
implicit method
clear
clc
format long g

%Defining Initial Conditions and Constants
Diffusion = 1; %Diffusion coefficient
Length = 1; %Length of the space
TempL = 1; %Temp at end of space
Temp0 = 5; %Temp at beginning of space
N = 1; %Constant factor for dimensionless
variables
n = 5; %Different values for nonlinear solution
space = 100; %Spatial resolution
x = 0:Length/space:Length; %Number of spatial points
xbar = x/Length; %Unitless Space
dx = Length/space; %Spatial resolution
dxbar = dx/Length; %Unitless Space
time = 1000; %Number of time steps
tconstant = 1000; %Number of time constants that
elapse
T = (Length.^2*N)/(Diffusion*Temp0.^n); %Time constant
tbar = (1000/tconstant)/(T); %Unitless time
dt = dx.^2/2; %Time resolution
dtbar = dt/tbar; %Unitless time resolution

```

```

w = 0.01;                                %Frequency of boundary
temperature
Temp = zeros(time,length(x)); %Size of temperature matrix
lambda = dtbar/((n+1)*dxbar.^2);

%Initial Conditions-----
-----
%Solving for the initial Temperature distribution
for j = 1:length(x)
    Temp(1,j) = ((1/Length)*(TempL.^(n+1)-
TempO.^(n+1))*xbar(j)...
    +TempO.^(n+1)).^(1/(n+1));
end

%Putting in the boundary conditions for all time
for i = 2:time
    %Temp(i,1) = TempO - time*dtbar*i;
    Temp(i,1) = TempO + sin(w*i);
    %Temp(i,1) = 0;
    Temp(i,length(x)) = TempL;
end

%Scaling Temperature to TempO
Temp = Temp/TempO;

%Nonlinear LU Decomposition-----
-----
%Note: boundary nodes are ignored for this computation.
Decompositions
%starts at j = 2 and ends at j = length(x)-1

%Filling Three Vectors of the Tridiagonal Matrix
d = zeros(size(x));
u = zeros(size(x));
l = zeros(size(x));
%Note: u(space-1)=0 and l(1) = 0

D = zeros(size(d)); %Intermediate vector for LU
decomposition
deltaTemp = zeros(size(Temp));
deltaTemp(:,1) = 0;
deltaTemp(:,length(x)) = 0;

h = waitbar(0,'There is no spoon...');
for m = 2:time
    waitbar(m/time)
    for i = 1:space-1

```

```

        %Setting up B
        B(m-1,i) = lambda*(Temp(m-1,i)).^(n+1) -
2*lambda*(Temp(m-1,i+1)).^(n+1)+lambda*(Temp(m-
1,i+2)).^(n+1);

        %Setting up the three diagonal vectors
        d(i) = (1+2*(n+1)*lambda*(Temp(m-1,i)).^n);
        if (i == 1)
            l(i) = 0;
        else
            l(i) = -1*lambda*(n+1)*(Temp(m-1,i-1)).^n;
        end
        if(i == space-1)
            u(i) = 0;
        else
            u(i) = -1*lambda*(n+1)*(Temp(m-1,i+1)).^n;
        end

        %Decomposition of coefficient matrix
        factor = l(i+1)/d(i);
        l(i+1) = factor;
        d(i+1) = d(i+1) - factor*u(i);

        %Forward Substitution
        if(i == 1)
            D(i) = B(m-1,i);
        else
            D(i) = B(m-1,i) - l(i)*D(i-1);
        end
    end
    for i = space:-1:2
        %Backward substitution
        if(i == space)
            deltaTemp(m-1,i) = D(i-1)/d(i-1);
        else
            deltaTemp(m-1,i) = (D(i-1) - u(i-
1)*deltaTemp(m-1,i+1))/d(i-1);
        end
    end

    %Solving for the new temperatures
    for i = 2:space
        Temp(m,i) = Temp(m-1,i) + deltaTemp(m-1,i);
    end
end
close(h);

```

```

figure(1)
plot(x,Temp(1,:),x,Temp(10,:),x,Temp(100,:),x,Temp(500,:),x
,Temp(1000,:));
xlabel('Length');
ylabel('Temperature');
title('Solution of Nonlinear Diffusion Equation');
legend('t = 1','t = 10','t = 100','t = 500','t = 1000');

```

B.6 HomoNonlinearDiffusion.m

```

%Created by Dustin Sipka
%EP 700
%Homogeneous Nonlinear Diffusion Equation
%September 8, 2005
%Plots the solution to the general homogeneous nonlinear
diffusion equation
%for varying values of n

clear
clc
format long g

%Defining Initial Conditions and Constants
Length = 1;      %Length of the space
TempL = 1;      %Temp at end of space
TempO = 5;      %Temp at beginning of space
n = [-5 -4 -3 -2 0 1 2 3 4 5]; %Different values for
nonlinear solution
x = 0:Length/100:Length; %Number of spatial points

%Solving for the Temperature distribution
%Note: i = power of the solution and j = spatial
temperature
h = waitbar(0,'There is no spoon...');
for i = 1:length(n)
    waitbar(i/length(n))
    for j = 1:length(x)
        Temp(i,j) = ((1/Length)*(TempL.^(n(i)+1)-
TempO.^(n(i)+1))*x(j)+TempO.^(n(i)+1)).^(1/(n(i)+1));
    end
end
close(h);

%Scaling Temperature to TempO

```

```

Temp = Temp/Temp0;

%Plotting the Different Solutions
plot(x,Temp)
xlabel('Length');
ylabel('Temperature');
title('Steady State Nonlinear Diffusion Equation for
Varying Powers');
legend('n = -5','n = -4','n = -3','n = -2','n =
0','n=1','n=2','n=3','n=4','n=5');

```

B.7 InitialConditionofInterest.m

```

%Created by Dustin Sipka
%EP 700
%Time Dependent Nonlinear Diffusion Equation
%January 30, 2006
%Solves the time dependent nonlinear equation using the
implicit method and
%lets it run to get the steady state distribution and
compare it to the
%theoretical prediction, also puts into one file all of the
analysis that
%has been done so results can be arrived at easily

clear
clc
format long g

%Defining Initial Conditions and Constants
%Parameters that can be controlled and changed
parameter = 1/2; %Parameter for Crank-
Nicolson variation
%Diffusion = 1; %Diffusion coefficient
Length = 1; %Length of the space
%TempL = 1; %Temp at end of space
%Temp0 = 1; %Temp at beginning of space
SteadyL = 0; %Temp at end for steady-
state solution
Steady0 = 1; %Temp at beginning of
steady-state solution
sLength = 0.5; %Length of steady-state
solution for constant source
sLength2 = 1; %Length of steady-state
solution with constant null

```



```

%N = 1; %Constant factor for
dimensionless variables (decrease N decrease penetration
depth, increase N increase penetration depth)
n = [0 1 2 3 4 5]; %Different
values for nonlinear solution
n2 = 0;
space = 100; %Spatial resolution
%w = 0.01*pi; %Frequency of source
time = 40000; %Number of time steps
dtbar = 1e-5; %Resolution of time

%Parameters the algorithm needs to know
sx = 0:sLength/(space):sLength; %Spatial
resolution for steady-state
sxbar = sx/sLength; %Unitless
space for steady-state
x = 0:Length/space:Length; %Number of
spatial points
xbar = x/Length; %Unitless
space
%dx = Length/space; %Spatial
resolution
dxbar = Length/space; %Unitless
space
%T = (Length.^2*N*(n+1))/(Diffusion*Temp0.^n); %Time
constant
%dtbar = dt/T; %Unitless
time resolution
tbar = time*dtbar; %Unitless
time
tbarv = 0:dtbar:tbar; %Unitless
time vector

%Parameters for analysis
Temp = zeros((time/10),length(x)); %Size of
temperature matrix
TempF = zeros(10,length(x)); %Size of
algorithm matrix
lambda = dtbar/(dxbar.^2); %Coefficient in
numerical algorithm
%Q = zeros(time,space+1); %Source Term
%Q(1,space/2) = 100; %Source Term
%periodtimesteps = round(2*pi/(w)); %Number of
time steps in one period of oscillation
t = 1:1:(time/10); %Time step
vector
t = t*10*dtbar; %tbar vector

```

```

%Steady0 = (SteadyL.^(n+1) + (Length-
0.5).*((Q(1,space/2)*dxbar*(n+1))/(2*Diffusion))).^(1/(n+1)
); %Temp at source for steady-state solution

for z = 1:length(n)
%Initial Conditions-----
-----
%Solving for the initial Temperature distribution
%Calculates the initial temperature distribution
for j = 1:length(sx)
    Steady1(j) = ((1/sLength)*(SteadyL.^(n(z)+1)-
Steady0.^(n(z)+1))*sxbar(j)...
    +Steady0.^(n(z)+1)).^(1/(n(z)+1));
%    Steady1(j) = 0;
end

%Calculates the steady-state for distribution with constant
null
for j = 1:length(sx)
    SteadySS(j) = ((1/sLength2)*(SteadyL.^(n(z)+1)-
Steady0.^(n(z)+1))*sxbar(j)...
    +Steady0.^(n(z)+1)).^(1/(n(z)+1));
end

%For distribution with extended null area
for i = 1:((space/2)+1)
    Steady3(i) = Steady1(i);
    Steady3(i+(space/2)) = SteadyL;
end

Temp(1,:) = Steady3;
TempSteady(z,:) = Temp(1,:);
end

figure(1)
plot(x,TempSteady);
xlabel('xbar');
ylabel('Temp');
legend('n=0','n=1','n=2','n=3','n=4','n=5');
title('Initial Conditions of Interest');

```

B.8 SourceGoodTimeNonlinearDiffusion.m

```

%Created by Dustin Sipka
%EP 700

```

```

%Time Dependent Nonlinear Diffusion Equation
%September 23, 2005
%Solves the time dependent nonlinear equation using the
implicit method
clear
clc
format long g

%Defining Initial Conditions and Constants
Diffusion = 1; %Diffusion coefficient
Length = 1; %Length of the space
TempL = 5; %Temp at end of space
Temp0 = 5; %Temp at beginning of space
N = 1; %Constant factor for dimensionless
variables
n = 3; %Different values for nonlinear solution
space = 100; %Spatial resolution
x = 0:Length/space:Length; %Number of spatial points
xbar = x/Length; %Unitless Space
dx = Length/space; %Spatial resolution
dxbar = dx/Length; %Unitless Space
time = 1000; %Number of time steps
tconstant = 1000; %Number of time constants that
elapse
T = (Length.^2*N)/(Diffusion*Temp0.^n); %Time constant
tbar = (1000/tconstant)/(T); %Unitless time
dt = dx.^2/2; %Time resolution
dtbar = dt/tbar; %Unitless time resolution
w = 0.01*pi; %Frequency of boundary
temperature
Temp = zeros(time,length(x)); %Size of temperature matrix
lambda = dtbar/((n+1)*dxbar.^2);
Q = zeros(time,space+1); %Source Term

%Initial Conditions-----
-----
% %Solving for the initial Temperature distribution
% for j = 1:length(x)
% Temp(1,j) = ((1/Length)*(TempL.^(n+1) -
Temp0.^(n+1))*xbar(j)...
% +Temp0.^(n+1)).^(1/(n+1));
% end
Temp(1,:) = 5;
Temp(1,50) = 10;
Temp(1,51) = 10;
Temp(1,52) = 10;

```

```

%Putting in the boundary conditions for all time
for i = 2:time
    %Temp(i,1) = Temp0 - time*dtbar*i;
    %Temp(i,1) = Temp0 + sin(w*i);
    Temp(i,1) = Temp0;
    Temp(i,length(x)) = TempL;
    %Q(i,space/2) = 2000;           %Steady source
    %Q(i,space/2) = 2000*sin(w*(i-1)); %Oscillating
source
end

%Scaling Temperature to Temp0
Temp = Temp/Temp0;

%Nonlinear LU Decomposition-----
-----
%Note: boundary nodes are ignored for this computation.
Decompositions
%starts at j = 2 and ends at j = length(x)-1

%Filling Three Vectors of the Tridiagonal Matrix
d = zeros(size(x));
u = zeros(size(x));
l = zeros(size(x));
%Note: u(space-1)=0 and l(1) = 0

D = zeros(size(d));           %Intermediate vector for LU
decomposition
deltaTemp = zeros(size(Temp));
deltaTemp(:,1) = 0;
deltaTemp(:,length(x)) = 0;

h = waitbar(0,'There is no spoon...');
for m = 2:time
    waitbar(m/time)
    for i = 1:space-1

        %Setting up B
        B(m-1,i) = lambda*(Temp(m-1,i)).^(n+1)-
2*lambda*(Temp(m-1,i+1)).^(n+1)+lambda*(Temp(m-
1,i+2)).^(n+1) + dtbar*Q(m-1,i);

        %Setting up the three diagonal vectors
        d(i) = (1+2*(n+1)*lambda*(Temp(m-1,i)).^n);
        if (i == 1)
            l(i) = 0;
        else

```

```

        l(i) = -1*lambda*(n+1)*(Temp(m-1,i-1)).^n;
    end
    if(i == space-1)
        u(i) = 0;
    else
        u(i) = -1*lambda*(n+1)*(Temp(m-1,i+1)).^n;
    end

    %Decomposition of coefficient matrix
    factor = l(i+1)/d(i);
    l(i+1) = factor;
    d(i+1) = d(i+1) - factor*u(i);

    %Forward Substitution
    if(i == 1)
        D(i) = B(m-1,i);
    else
        D(i) = B(m-1,i) - l(i)*D(i-1);
    end
end
for i = space:-1:2
    %Backward substitution
    if(i == space)
        deltaTemp(m-1,i) = D(i-1)/d(i-1);
    else
        deltaTemp(m-1,i) = (D(i-1) - u(i-
1)*deltaTemp(m-1,i+1))/d(i-1);
    end
end

    %Solving for the new temperatures
    for i = 2:space
        Temp(m,i) = Temp(m-1,i) + deltaTemp(m-1,i);
    end
end
close(h);

%Plots the Different Time Steps
%mov = avifile('difmov.avi','fps',10);
for i = 1:time/10
    figure(1);
    plot(x,Temp(10*i,:));
    title('General Nonlinear Diffusion with Non-Zero
Boundary Conditions');
    xlabel('Length of Wave Solution');
    ylabel('Amplitude of Solution');
    %legend('Numerical Solution','Exact Solution');

```

```

    axis([0 Length 0 2]);
    F(i) = getframe;
    %mov = addframe(mov,F(i));
end
%mov = close(mov);
movie(F);    %Shows Plots as a Movie

```

B.9 TimeConstantsNonlinear.m

```

%Created by Dustin Sipka
%EP 700
%Time Dependent Nonlinear Diffusion Equation
%March 2, 2006
%Calculates the time-constant function and compares to
actual time-constants
%calculated using FinalCrankNicolsonAlgorithm.m

clear
clc
format long g

%Defining Initial Conditions and Constants
%Parameters that can be controlled and changed
parameter = 1/2;           %Parameter for Crank-
Nicolson variation
%Diffusion = 1;           %Diffusion coefficient
Length = 1;               %Length of the space
%TempL = 1;               %Temp at end of space
%TempO = 1;               %Temp at beginning of space
SteadyL = 0;              %Temp at end for steady-
state solution
SteadyO = 1;              %Temp at beginning of
steady-state solution
sLength = 0.5;            %Length of steady-state
solution for constant source
sLength2 = 1;             %Length of steady-state
solution with constant null
%N = 1;                   %Constant factor for
dimensionless variables (decrease N decrease penetration
depth, increase N increase penetration depth)
n = 0:0.1:10;             %Different values for
nonlinear solution
n2 = 0;
space = 200;              %Spatial resolution
%w = 0.01*pi;             %Frequency of source

```

```

time = 40000;                %Number of time steps
dtbar = 1e-5;               %Resolution of time

%Parameters the algorithm needs to know
sx = 0:sLength/(space):sLength;           %Spatial
resolution for steady-state
sxbar = sx/sLength;                       %Unitless
space for steady-state
x = 0:Length/space:Length;                %Number of
spatial points
xbar = x/Length;                           %Unitless
space
%dx = Length/space;                       %Spatial
resolution
dxbar = Length/space;                     %Unitless
space
%T = (Length.^2*N*(n+1))/(Diffusion*Temp0.^n); %Time
constant
%dtbar = dt/T;                             %Unitless
time resolution
tbar = time*dtbar;                         %Unitless
time
tbarv = 0:dtbar:tbar;                     %Unitless
time vector

%Calculates the steady-state for distribution with constant
null
for k = 1:length(n)
    for j = 1:length(sx)
        SteadySS(k,j) = ((1/sLength2)*(SteadyL.^(n(k)+1)-
SteadyO.^(n(k)+1))*sxbar(j)...
        +SteadyO.^(n(k)+1)).^(1/(n(k)+1));
        Tavg(k) = sum(SteadySS(k,:))/(space+1);
    end
    Time-constantfunction(k) = 1/((n(k)+1)*Tavg(k).^n(k));
end

%Calculates the steady-state for distribution with constant
null
for k = 1:length(n)
    for j = 1:length(sx)
        SteadySS2(k,j) = ((1/sLength2)*(SteadyL.^(n(k)+1)-
SteadyO.^(n(k)+1))*sxbar(j)...
        +SteadyO.^(n(k)+1)).^(1/(n(k)+1));
        Tavg2(k) = sum(SteadySS(k,).^n(k))/(space+1);
    end
    Time-constantfunction2(k) = 1/((n(k)+1)*Tavg(k));
end

```

```

end

figure(1)
plot(n,Time-constantfunction,n,Time-
constantfunction2,0,pi^2*0.112307,'o',1,pi^2*0.086960,'o',2
,pi^2*0.068575,'o',3,pi^2*0.056119,'o',4,pi^2*0.047294,'o',5
,pi^2*0.040750,'o',6,pi^2*0.035710,'o',7,pi^2*0.031710,'o'
,8,pi^2*0.028454,'o',9,pi^2*0.025751,'o',10,pi^2*0.023468,'
o');
xlabel('n');
ylabel('pi^2*tau');
legend('<T>^n','<T^>n');
title('Theoretical Prediction of Timescale Compared to
Actual Values');

figure(2)
plot(n,Time-
constantfunction,0,pi^2*0.112307,'o',1,pi^2*0.086960,'o',2,
pi^2*0.068575,'o',3,pi^2*0.056119,'o',4,pi^2*0.047294,'o',5
,pi^2*0.040750,'o',6,pi^2*0.035710,'o',7,pi^2*0.031710,'o',
8,pi^2*0.028454,'o',9,pi^2*0.025751,'o',10,pi^2*0.023468,'o
');
xlabel('n');
ylabel('pi^2*tau');

```

B.10 VaryingParameter.m

```

%Created by Dustin Sipka
%EP 700
%General Diffusion Equation
%Crank-Nicolson Method
%April 25, 2005
%Varying the Parameter for implicit and explicit

clear
clc
format long g

%Note that in T(i,n) "i" represents time and "n" represents
space

%Definition of constants and parameters
parameter = 0:0.01:1;           %Determines how much implicit
and explicit are used
order = 5;                       %Order of Interest

```



```

diffusion = 1;           %Diffusion Coefficient
L = 1;                 %Length of Wave
T = L.^2/diffusion;    %Makes Coefficient "1"
time = 100;           %Time Steps
tconstant = 10;        %Number of time constants that
elapse
tbar = (time/tconstant)/T; %Unitless Time
Steps
space = 100;           %Resolution of Solution
dx = L/space;          %Spatial resolution
dxbar = dx/L;          %Unitless Spatial Resolution
dt = ((L/(order*pi)).^2/diffusion); %Time
resolution
dtbar = dt/tbar;       %Unitless dt
theta = 0:L/space:L;   %Values for Solution
xbar = theta/L;        %Unitless Space
lambda = (diffusion*dtbar)/(dxbar.^2); %Constant from
FDE
Temp = zeros(time,space+1);
h = waitbar(0,'How long is this gonna take...');

%Loop that will vary the parameter for mixing explicit and
implicit
for b = 1:length(parameter)
waitbar(b/length(parameter));
alpha = 1+2*lambda*parameter(b); %Constant from FDE

%Solving Numerically Using LU Decomposition-----
-----
%Boundary Conditions
for i = 1:time
    Temp(i,1) = 0;
    Temp(i,space+1) = 0;
end

%Initial Conditions (b vector in LU Decompostion)
Temp(1,:) = sin(pi*xbar) + (1/1)*sin(2*pi*xbar)...
    + (1/1)*sin(3*pi*xbar) + (1/1)*sin(4*pi*xbar)...
    + (1/1)*sin(5*pi*xbar) + (1/1)*sin(6*pi*xbar)...
    + (1/1)*sin(7*pi*xbar) + (1/1)*sin(8*pi*xbar)...
    + (1/1)*sin(9*pi*xbar) + (1/1)*sin(10*pi*xbar);

%Filling Three Vectors of the Tridiagonal Matrix
for n = 1:space-1
    d(n) = alpha;
    u(n) = -lambda*parameter(b);
    l(n) = -lambda*parameter(b);

```

```

end
u(space-1) = 0;
l(1) = 0;
D = zeros(size(d)); %Intermediate Vector in LU
Decomposition

%Decomposition of Tridiagonal Matrix
for k = 2:space-1
    factor = l(k)/d(k-1);
    l(k) = factor;
    d(k) = d(k) - factor*u(k-1);
end

%Forward and Backward Substitution to Solve for Next Time
Step
for m = 2:time
    B = Temp;
    for q = 2:space-1
        B(m-1,q) = B(m-1,q) + (lambda-
lambda*parameter(b))*(Temp(m-1,q+1)-2*Temp(m-1,q)+Temp(m-
1,q-1));
    end
    %Forward (only works for tridiagonal sparse matrix)
    for i = 1:space-1
        if (i == 1)
            B(m-1,i+1) = B(m-1,i+1) +
lambda*parameter(b)*B(m-1,i);
        elseif (i == space-1)
            B(m-1,i+1) = B(m-1,i+1) +
lambda*parameter(b)*B(m-1,i+2);
        end
        if (i == 1)
            D(i) = B(m-1,i+1); %First element of D
        else
            D(i) = B(m-1,i+1) - l(i)*D(i-1); %Forward
loop
        end
    end
    %Backward (only works for tridiagonal sparse matrix)
    for i = space:-1:2
        if (i == space)
            Temp(m,i) = D(i-1)/d(i-1); %Last element of X
        else
            Temp(m,i) = (D(i-1) - u(i-1)*Temp(m,i+1))/d(i-
1); %Backward loop
        end
    end
end

```

```

end

%Performs Fourier Transform to Find Amplitudes of Different
Modes-----
for i = 1:time
    for m = 1:10
        for n = 1:space
            transform(n) =
Temp(i,n)*sin(m*pi*xbar(n))/(space);
        end
        Amplitude(i,m) = 2*sum(transform);
    end
end

end

%Solving the Exact Solution -----
-----
%Setting Up Time Constant
for n = 1:10
    tau(n) = (L/((n)*pi)).^2/diffusion;
end

dtexact = tau(order)/(time/tconstant);
texact = 0:dtexact:time*dtexact;

%Setting Up Amplitudes
for i = 1:10
    for n = 1:time
        A(n,i) = exp(-texact(n)/tau(i));
    end
end

end

%Exact Solution
%Boundary Conditions
for i = 1:time
    Ex(i,1) = 0;
    Ex(i,space+1) = 0;
end

end

%Solution
for i = 1:time
    for n = 2:space
        Ex(i,n) = A(i,1)*sin(pi*xbar(n)) +
A(i,2)*sin(2*pi*xbar(n))...
        + A(i,3)*sin(3*pi*xbar(n)) +
A(i,4)*sin(4*pi*xbar(n))...
        + A(i,5)*sin(5*pi*xbar(n)) +
A(i,6)*sin(6*pi*xbar(n))...
    end
end

```

```

        + A(i,7)*sin(7*pi*xbar(n)) +
A(i,8)*sin(8*pi*xbar(n))...
        + A(i,9)*sin(9*pi*xbar(n)) +
A(i,10)*sin(10*pi*xbar(n));
    end
end

%-----
%Calculates Error Using RMS of Actual Solution to the
Numerical Solution of
%'order'
ErrorSol = (Temp - Ex).^2;
for i = 1:time
    RMS(i) = sqrt(sum(ErrorSol(i,:))/(space+1));
end
RMSP(b) = RMS(time);

%Calculates Error Between Actual and Numerical Amplitudes
ErrorAmp = abs(A(:,order) - Amplitude(:,order));
end
close(h);

%Plotting-----
-----
x = 1:1:time; %Used to Plot Change in Error with Time

figure(6)
plot(theta,Temp(1,:),theta,Temp(25,:), '--
',theta,Temp(50,:), ':',theta,Temp(75,:), '-. ');
ylabel('T');
xlabel('Length of Solution');
title('Solution of General Diffusion Equation');
legend('tbar = 0','tbar = 25', 'tbar = 50', 'tbar = 75');

figure(7)
semilogy(parameter,RMSP);
title('RMS Error Between Numerical and Exact Solutions');
xlabel('Parameter Value');
ylabel('RMS Error')

[C,I] = min(RMSP);
parameter = parameter(I)

```

B.11 VaryingParameter2.m

```
%Created by Dustin Sipka
%EP 700
%General Diffusion Equation
%Crank-Nicolson Method
%April 25, 2005
%Varying the Parameter for implicit and explicit

clear
clc
format long g

%Note that in T(i,n) "i" represents time and "n" represents
space

%Definition of constants and parameters
parameter = 0:0.01:1; %Determines how much implicit and
explicit are used
order = 6; %Order of Interest
diffusion = 1; %Diffusion Coefficient
L = 1; %Length of Wave
T = L.^2/diffusion; %Makes Coefficient "1"
time = 100; %Time Steps
tconstant = 1; %Number of time constants that
elapse
tbar = (time/tconstant)/T; %Unitless Time
Steps
space = 300; %Resolution of Solution
dx = L/space; %Spatial resolution
dxbar = dx/L; %Unitless Spatial Resolution
%dt = ((L/(order*pi)).^2/diffusion); %Time
resolution
dt = 0.0005:0.0001:0.003; %Time resolution
dtbar = dt/tbar; %Unitless dt
theta = 0:L/space:L; %Values for Solution
xbar = theta/L; %Unitless Space
lambda = (diffusion*dtbar)/(dxbar.^2); %Constant from
FDE
Temp = zeros(time,space+1);
result = zeros(length(dt),3);
%h = waitbar(0,'How long is this gonna take...');

for z = 1:length(dt)
    %waitbar(z/length(dt));
%Loop that will vary the parameter for mixing explicit and
implicit
```

```

for b = 1:length(parameter)
%waitbar(b/length(parameter));
alpha = 1+2*lambda(z)*parameter(b);          %Constant from FDE

%Solving Numerically Using LU Decomposition-----
-----
%Boundary Conditions
for i = 1:time
    Temp(i,1) = 0;
    Temp(i,space+1) = 0;
end

%Initial Conditions (b vector in LU Decompostion)
Temp(1,:) = sin(pi*xbar) + (1/1)*sin(2*pi*xbar)...
    + (1/1)*sin(3*pi*xbar) + (1/1)*sin(4*pi*xbar)...
    + (1/1)*sin(5*pi*xbar) + (1/1)*sin(6*pi*xbar)...
    + (1/1)*sin(7*pi*xbar) + (1/1)*sin(8*pi*xbar)...
    + (1/1)*sin(9*pi*xbar) + (1/1)*sin(10*pi*xbar);

%Filling Three Vectors of the Tridiagonal Matrix
for n = 1:space-1
    d(n) = alpha;
    u(n) = -lambda(z)*parameter(b);
    l(n) = -lambda(z)*parameter(b);
end
u(space-1) = 0;
l(1) = 0;
D = zeros(size(d)); %Intermediate Vector in LU
Decomposition

%Decomposition of Tridiagonal Matrix
for k = 2:space-1
    factor = l(k)/d(k-1);
    l(k) = factor;
    d(k) = d(k) - factor*u(k-1);
end

%Forward and Backward Substitution to Solve for Next Time
Step
for m = 2:time
    B = Temp;
    for q = 2:space-1
        B(m-1,q) = B(m-1,q) + (lambda(z)-
lambda(z)*parameter(b))*(Temp(m-1,q+1)-2*Temp(m-
1,q)+Temp(m-1,q-1));
    end
    %Forward (only works for tridiagonal sparse matrix)

```

```

    for i = 1:space-1
        if (i == 1)
            B(m-1,i+1) = B(m-1,i+1) +
lambda(z)*parameter(b)*B(m-1,i);
        elseif (i == space-1)
            B(m-1,i+1) = B(m-1,i+1) +
lambda(z)*parameter(b)*B(m-1,i+2);
        end
        if (i == 1)
            D(i) = B(m-1,i+1); %First element of D
        else
            D(i) = B(m-1,i+1) - l(i)*D(i-1); %Forward
loop
        end
    end
    %Backward (only works for tridiagonal sparse matrix)
    for i = space:-1:2
        if (i == space)
            Temp(m,i) = D(i-1)/d(i-1); %Last element of X
        else
            Temp(m,i) = (D(i-1) - u(i-1)*Temp(m,i+1))/d(i-
1); %Backward loop
        end
    end
end
end

%Performs Fourier Transform to Find Amplitudes of Different
Modes-----
for i = 1:time
    for m = 1:10
        for n = 1:space
            transform(n) =
Temp(i,n)*sin(m*pi*xbar(n))/(space);
        end
        Amplitude(i,m) = 2*sum(transform);
    end
end
end

%Solving the Exact Solution -----
-----
%Setting Up Time Constant
for n = 1:10
    tau(n) = (L/((n)*pi)).^2/diffusion;
end

dtexact = tau(order)/(time/tconstant);
texact = 0:dtexact:time*dtexact;

```

```

%Setting Up Amplitudes
for i = 1:10
    for n = 1:time
        A(n,i) = exp(-texact(n)/tau(i));
    end
end

%Exact Solution
%Boundary Conditions
for i = 1:time
    Ex(i,1) = 0;
    Ex(i,space+1) = 0;
end

%Solution
for i = 1:time
    for n = 2:space
        Ex(i,n) = A(i,1)*sin(pi*xbar(n)) +
A(i,2)*sin(2*pi*xbar(n))...
            + A(i,3)*sin(3*pi*xbar(n)) +
A(i,4)*sin(4*pi*xbar(n))...
            + A(i,5)*sin(5*pi*xbar(n)) +
A(i,6)*sin(6*pi*xbar(n))...
            + A(i,7)*sin(7*pi*xbar(n)) +
A(i,8)*sin(8*pi*xbar(n))...
            + A(i,9)*sin(9*pi*xbar(n)) +
A(i,10)*sin(10*pi*xbar(n));
    end
end

%-----
-----
%Calculates Error Using RMS of Actual Solution to the
Numerical Solution of
%'order'
ErrorSol = (Temp - Ex).^2;
for i = 1:time
    RMS(i) = sqrt(sum(ErrorSol(i,:))/(space+1));
end
RMSP(b) = RMS(time);

%Calculates Error Between Actual and Numerical Amplitudes
ErrorAmp = abs(A(:,order) - Amplitude(:,order));
end
%close(h);

```



```

%Plotting-----
-----
xt = 1:1:time; %Used to Plot Change in Error with Time

figure(7)
semilogy(parameter,RMSP);
title('RMS Error Between Numerical and Exact Solutions');
xlabel('Parameter Value');
ylabel('RMS Error');
hold on

[C,I] = min(RMSP);
result(z,1) = parameter(I);
result(z,2) = RMSP(I);
result(z,3) = dt(z);
end
%close(h);

result

```

B.12 WavefrontVelocity.m

```

%Created by Dustin Sipka
%EP 700
%Time Dependent Nonlinear Diffusion Equation
%January 30, 2006
%Solves the time dependent nonlinear equation using the
implicit method and
%lets it run to get the steady state distribution and
compare it to the
%theoretical prediction, also puts into one file all of the
analysis that
%has been done so results can be arrived at easily

clear
clc
format long g

%Defining Initial Conditions and Constants
%Parameters that can be controlled and changed
parameter = 1/2; %Parameter for Crank-
Nicolson variation
%Diffusion = 1; %Diffusion coefficient
Length = 1; %Length of the space
%TempL = 1; %Temp at end of space

```

```

%Temp0 = 1; %Temp at beginning of space
SteadyL = 0; %Temp at end for steady-
state solution
Steady0 = 1; %Temp at beginning of
steady-state solution
sLength = 0.5; %Length of steady-state
solution for constant source
sLength2 = 1; %Length of steady-state
solution with constant null
%N = 1; %Constant factor for
dimensionless variables (decrease N decrease penetration
depth, increase N increase penetration depth)
n = 1:1:10; %Different values for
nonlinear solution
n2 = 0;
space = 300; %Spatial resolution
%w = 0.01*pi; %Frequency of source
time = 400000; %Number of time steps
dtbar = 1e-6; %Resolution of time

%Parameters the algorithm needs to know
sx = 0:sLength/(space):sLength; %Spatial
resolution for steady-state
sxbar = sx/sLength; %Unitless
space for steady-state
x = 0:Length/space:Length; %Number of
spatial points
xbar = x/Length; %Unitless
space
%dx = Length/space; %Spatial
resolution
dxbar = Length/space; %Unitless
space
%T = (Length.^2*N*(n+1))/(Diffusion*Temp0.^n); %Time
constant
%dtbar = dt/T; %Unitless
time resolution
tbar = time*dtbar; %Unitless
time
tbarv = 0:dtbar:tbar; %Unitless
time vector

%Parameters for analysis
Temp = zeros((time/10),length(x)); %Size of
temperature matrix
TempF = zeros(10,length(x)); %Size of
algorithm matrix

```

```

lambda = dtbar/(dxbar.^2); %Coefficient in
numerical algorithm
%Q = zeros(time,space+1); %Source Term
%Q(1,space/2) = 100; %Source Term
%periodtimesteps = round(2*pi/(w)); %Number of time
steps in one period of oscillation
t = 1:1:(time/10); %Time step
vector
t = t*10*dtbar; %tbar vector
%SteadyO = (SteadyL.^(n+1) + (Length-
0.5).*(Q(1,space/2)*dxbar*(n+1))/(2*Diffusion)).^(1/(n+1)
); %Temp at source for steady-state solution

for z = 1:length(n)

%Initial Conditions-----
-----
%Solving for the initial Temperature distribution
%Calculates the initial temperature distribution
for j = 1:length(sx)
    Steady1(j) = ((1/sLength)*(SteadyL.^(n(z)+1)-
SteadyO.^(n(z)+1))*sxbar(j)...
    +SteadyO.^(n(z)+1)).^(1/(n(z)+1));
% Steady1(j) = 0;
end

%Calculates the steady-state for distribution with constant
null
for j = 1:length(sx)
    SteadySS(j) = ((1/sLength2)*(SteadyL.^(n(z)+1)-
SteadyO.^(n(z)+1))*sxbar(j)...
    +SteadyO.^(n(z)+1)).^(1/(n(z)+1));
end

%For distribution with extended null area
for i = 1:((space/2)+1)
    Steady3(i) = Steady1(i);
    Steady3(i+(space/2)) = SteadyL;
end

Temp(1,:) = Steady3;
%Temp(1,1) = 1;

%Putting in the boundary conditions for all time
for i = 0:((time/10)-1)
    Temp(i+1,1) = Temp(1,1);
    Temp(i+1,length(x)) = Temp(1,length(x));

```

```

%      Q(i,space/2) = 100;          %Steady source
%      Q(i,(space/2)) = Q(i,space/2) + 100*sin(w*(i-1));
%Oscillating source with steady source
end

for i = 1:10
    TempF(i,1) = Temp(1,1);
    TempF(i,length(x)) = Temp(1,length(x));
end

%Nonlinear LU Decomposition-----
-----
%Note: boundary nodes are ignored for this computation.
Decompositions
%starts at j = 2 and ends at j = length(x)-1

%Filling Three Vectors of the Tridiagonal Matrix
d = zeros(size(x));
u = zeros(size(x));
l = zeros(size(x));
%Note:  u(space-1)=0 and l(1) = 0

D = zeros(size(d));          %Intermediate vector for LU
decomposition
deltaTemp = zeros(size(TempF));
deltaTemp(:,1) = 0;
deltaTemp(:,length(x)) = 0;

h = waitbar(0,'There is no spoon...');
for q = 1:((time/10)-1)
    waitbar(q/(time/10),h)
    TempF(1,:) = Temp(q,:);
    for m = 2:10
        for i = 1:space-1

            %Setting up B
            B(m-1,i) = lambda*parameter*(TempF(m-
1,i)).^(n(z)+1)-2*lambda*parameter*(TempF(m-
1,i+1)).^(n(z)+1)+lambda*parameter*(TempF(m-
1,i+2)).^(n(z)+1)+(1-parameter)*lambda*(TempF(m-
1,i)).^(n(z)+1)-2*TempF(m-1,i+1).^(n(z)+1)+TempF(m-
1,i+2).^(n(z)+1));% + dtbar*Q(m-1,i);

            %Setting up the three diagonal vectors
            d(i) = (1+2*(n(z)+1)*lambda*parameter*(TempF(m-
1,i)).^n(z));
            if (i == 1)

```

```

        l(i) = 0;
    else
        l(i) = -1*lambda*parameter*(n(z)+1)*(TempF(m-
1,i-1)).^n(z);
    end
    if(i == space-1)
        u(i) = 0;
    else
        u(i) = -1*lambda*parameter*(n(z)+1)*(TempF(m-
1,i+1)).^n(z);
    end

    %Decomposition of coefficient matrix
    factor = l(i+1)/d(i);
    l(i+1) = factor;
    d(i+1) = d(i+1) - factor*u(i);

    %Forward Substitution
    if(i == 1)
        D(i) = B(m-1,i);
    else
        D(i) = B(m-1,i) - l(i)*D(i-1);
    end
end
for i = space:-1:2
    %Backward substitution
    if(i == space)
        deltaTemp(m-1,i) = D(i-1)/d(i-1);
    else
        deltaTemp(m-1,i) = (D(i-1) - u(i-
1)*deltaTemp(m-1,i+1))/d(i-1);
    end
end

    %Solving for the new temperatures
    for i = 2:space
        TempF(m,i) = TempF(m-1,i) + deltaTemp(m-1,i);
    end
end
Temp(q+1,:) = TempF(10,:);
end
close(h);

[a,b] = find(Temp > 0.099 & Temp < 0.100);

figure(1)
hold on

```

```
plot(a*dtbar*10,b*dxbar);  
xlabel('Time tbar');  
ylabel('Position xbar');  
title('Position vs Time for Specific Point on Wavefront');  
end
```